Bachelorarbeit

OpenStreetMap in R - Freie Räumliche Daten für geostatistische Analysen

Schlesinger Thomas

Professor: Prof. Dr. Torsten Hothorn

Betreuung: Manuel Eugster

Institut für Statistik, Ludwig-Maximilians-Universität München
30.08.2011

Abstract

Im Rahmen der Bachelorarbeit „OpenStreetMap in R“ wurde ein **R**-Paket namens osmar geschrieben. Mit Hilfe dieses Paketes findet der Zugriff auf Daten der OpenStreetMap-Datenbank statt.

Innerhalb dieser Arbeit geht es darum, die zugrundeliegenden Datenstrukturen zu erläutern. Weiterhin wird ein Überblick über die wichtigsten Funktionen der Version `1.0` mit zwei möglichen Anwendungen gegeben.

Inhaltsverzeichnis

1 Einleitung

OpenStreetMap ist ein Community-Projekt, welches oft auch als „Wikipedia der Kartographie“ bezeichnet wird: Sämtliche geographischen Daten sind von Usern erstellt bzw. können geändert werden und jede Person hat Online Zugriff auf die Karten bzw. Datenbanken. Da es sich dabei um Daten verschiedenster Art handelt, können diese auch für statistische Analysen verwendet werden.

Das Werkzeug des modernen Statistikers ist das freie Programmpaket **R** (R Development Core Team, 2011), welches auf der Programmiersprache *S* basiert. Einer der Vorteile dieses Programmes liegt darin, es mit eigenen oder von anderen Usern geschriebenen Paketen zu erweitern. Ein Paket, welches zur Bearbeitung geographischer Daten geeignet ist, ist das sp-Paket (Pebesma and Bivand, 2011).

Im Rahmen dieser Bachelorarbeit *„OpenStreetMap in R“* geht es darum, die Daten aus der *OpenStreetMap*-Datenbank abzurufen und mit Hilfe des sp-Paketes ein Paket (namens osmar) für **R** zu schreiben. Dieses stellt die Daten für die Bearbeitung in **R** und erste Methoden zur Analyse dieser Daten zur Verfügung.

Als Erstes folgt eine Einführung in das *OpenStreetMap*-Projekt (Kapitel 2). Diese besteht darin zu erklären, was *OpenStreetMap* ist, wie die Webpräsenz aussieht, worin genau die Lizenz besteht und wo sämtliche Daten herkommen.

In Kapitel 3 geht es um die beiden vorliegenden Datenstrukturen. Einmal das Datenmodell, welches *OpenStreetMap* zu Grunde liegt, und zweitens die Datenstruktur, die das sp-Paket zur Verfügung stellt (Kapitel 3.1 & 3.2). Die Verbindung dieser Strukturen folgt in Kapitel 3.3.

Die Ausführung in **R** und zwei Anwendungsbeispiele folgen in den nächsten beiden Kapiteln. Als Erstes werden die Grundfunktionen des Paketes beschrieben (Kapitel 4). Mit Hilfe dieser folgt ein komplexeres Beispiel, in der die wichtigsten Funktionen vorkommen und angewendet werden (Kapitel 5).

Das letzte Kapitel (Kapitel 6) der Bachelorarbeit handelt davon, was in Zukunft mit dem Paket noch passiert bzw. passieren kann.

2 Einführung in OpenStreetMap

Dieses Kapitel gibt grundlegende Informationen über *OpenStreetMap*, stellt das Webinterface vor, zeigt den aktuellen Stand über die Lizenz und schildert die Entstehung der Daten.

2.1 Was ist *OpenStreetMap*?

OpenStreetMap ist ein Community-Projekt, welches im Jahr 2004 in Großbritannien von Steve Coast gestartet wurde (Ramm and Topf, 2010, S.1). Erreichbar ist es unter `www.openstreetmap.org`. Ähnlich dem Projekt *Wikipedia* (`www.wikipedia.org`) handelt es davon, von Nutzern bereitgestellte oder gesammelte Daten zusammenzutragen und sie anderen Nutzern frei zur Verfügung zu stellen.

Der Unterschied in *OpenStreetMap* besteht allerdings darin, dass die Daten vor allem geographischer Natur sind. Personen laufen im simpelsten Fall mit GPS-Geräten Orte ab und tragen sie in die Datenbank ein (siehe Kapitel 2.4). Anschließend werden diese Daten mit Programmen in Karten umgewandelt und zur Verfügung gestellt. Die Problematik darin ist klar ersichtlich, denn es kann durchaus sehr laienhaft sein und qualitative Mängel können auftreten. Aber gerade deshalb ist es sehr bemerkenswert, wenn man sich die bisherige Leistung anschaut.

Laut dem Statistikreport vom 13. Juli 2011 (siehe OpenStreetMap, 2011d) sind 432.463 User angemeldet und es wurden 2.412.074.192 GPS-Points hochgeladen. Wobei im Allgemeinen davon ausgegangen wird, dass nur etwa 10% der User aktiv und 1% der User so genannte Heavy-User sind. Dies ist aber bei den meisten Open-Community-Projekten der Fall (Nielson, 2006).

Die Abdeckung der Karten umfasst mittlerweile die meisten größeren Städte und wichtigsten Straßen (ca. 33 Millionen Kilometer Straße, Stand: Mai 2009)(OpenStreetMap, 2011c). Allerdings ist es bei einem Projekt dieser Art auch nicht möglich „fertig“ zu sein, denn Straßen ändern sich und es können immer noch weitere Daten hinzugefügt werden.

Wenn man zum Vergleich mit Wikipedia zurückkehrt, fallen einem weitere Gemeinsamkeiten auf. Es werden zum Beispiel keine festen Strukturen vorgegeben, in denen die Daten angelegt werden müssen. Wobei mittlerweile ein Konsens über die häufigsten Einträge besteht. Durch eine hohe Änderungsrate in einigen Gebieten wird das Nachvollziehen der Änderungen interessant, so dass jede Veränderung gespeichert wird um dies nachzuvollziehen. Zu den „negativen“ Gemeinsamkeiten zählt die Qualität der Daten. Diese sind nämlich nur so gut, wie der Benutzer, der sie verfasst. (Ramm and Topf, 2010, S.2)

Unterschiede zu Wikipedia gibt es allerdings ebenso viele. Die erwähnte Änderungshistorie ist zwar simpel nachzuschauen, aber Änderungen wieder rückgängig zu machen (z.B. ältere Versionen von Karten anzuschauen) ist komplizierter als bei Wikipedia. Dies liegt vor allem auch an der flacheren Lernkurve des Editors, denn bei *OpenStreetMap* ist eine höhere Einarbeitungszeit von Nöten. (Ramm and Topf, 2010, S.2)

Weitere Vergleiche lassen sich auch mit anderen Onlinekartendiensten ziehen. Diese basieren wie *OpenStreetMap* auf Datenbanken mit Geodaten. Der Unterschied besteht

allerdings darin, dass man bei *OpenStreetMap* direkt auf die geographischen Daten zugreifen kann. Aus diesen kann man die Karten zeichnen. Bei anderen Diensten werden nur die Karten zur Verfügung gestellt und der umgekehrte Weg zurück zu den Daten ist nicht möglich. (Ramm and Topf, 2010, S.5f)

2.2 Webpräsenz

Die Website (erreichbar unter `www.openstreetmap.org`) ist die erste Anlaufstelle für Personen, die Interesse an *OpenStreetMap* zeigen. Auf Abb.1 (S.10) ist das Grundlayout zu erkennen. Prinzipiell sieht es aus, wie andere Kartendienste im Internet, z.B. `maps.google.de`, `de.maps.yahoo.com` oder `www.bing.com/maps`. Allerdings mit einem anderen Layout der Karten.

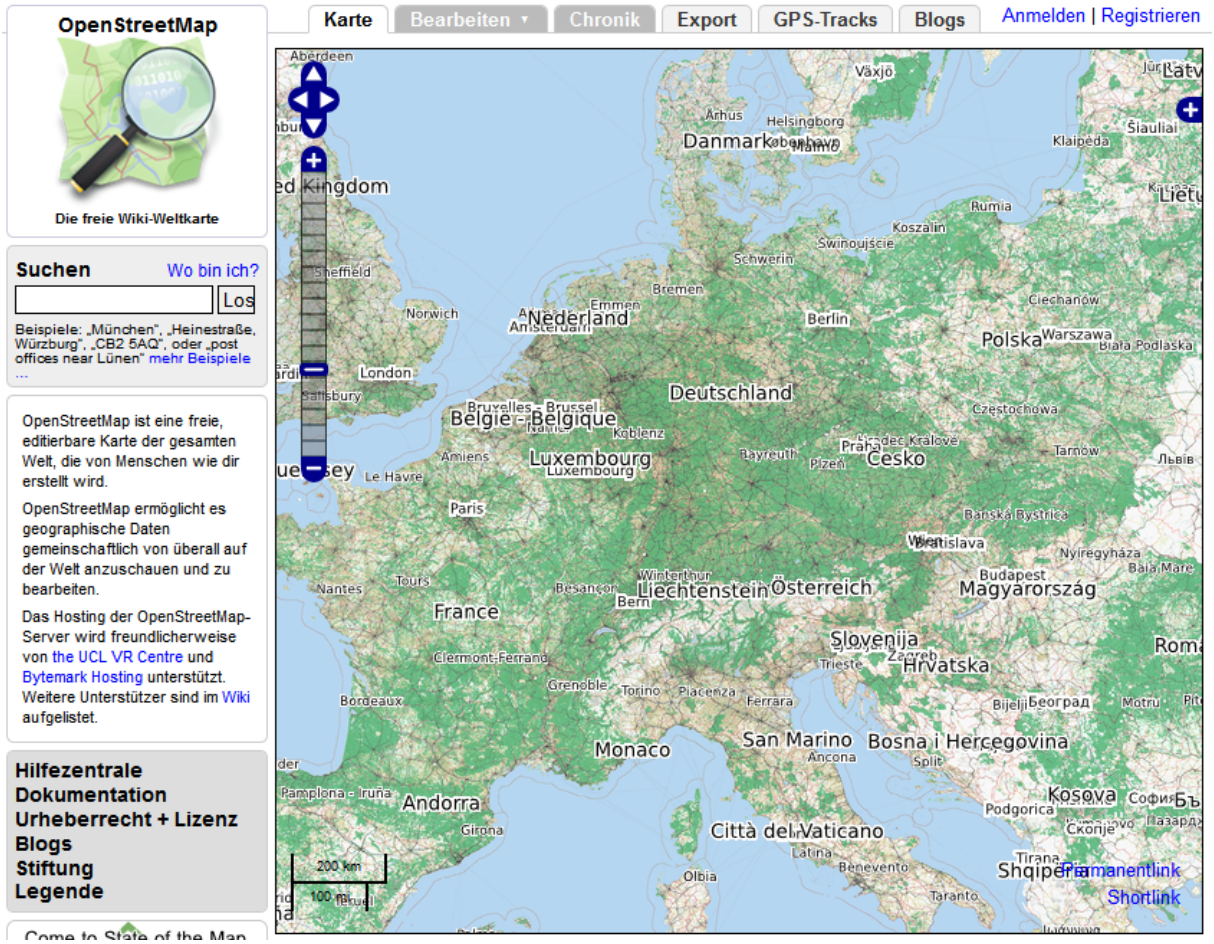


Abbildung 1: *OpenStreetMap* beim Aufrufen der Website

Auf der linken Seite erkennt man ein standardmäßiges Suchfeld und verschiedene Links zu Seiten mit weiteren Informationen. Die bei den oben genannten Seiten implementierte

Routenplanung ist standardmäßig noch nicht in *OpenStreetMap* implementiert. Hierzu gibt es allerdings schon Projekte (weiteres dazu später).

Interessant wird es, wenn man rechts oben auf das +-Symbol klickt (siehe Abb.2, S.11). Zur Auswahl stehen 2 Mapstyles (Mapnik und Osmarender) und 2 unterschiedliche Filter, die man über die Karte legen kann (Radfahrerkarte und Straßen ohne Name). Außerdem ist bei weiterem Heranzoomen im Vergleich mit GoogleMaps erkennbar, dass unter Umständen bei *OpenStreetMap* mehr Einrichtungen eingezeichnet sind (siehe Abb.3, S.12).

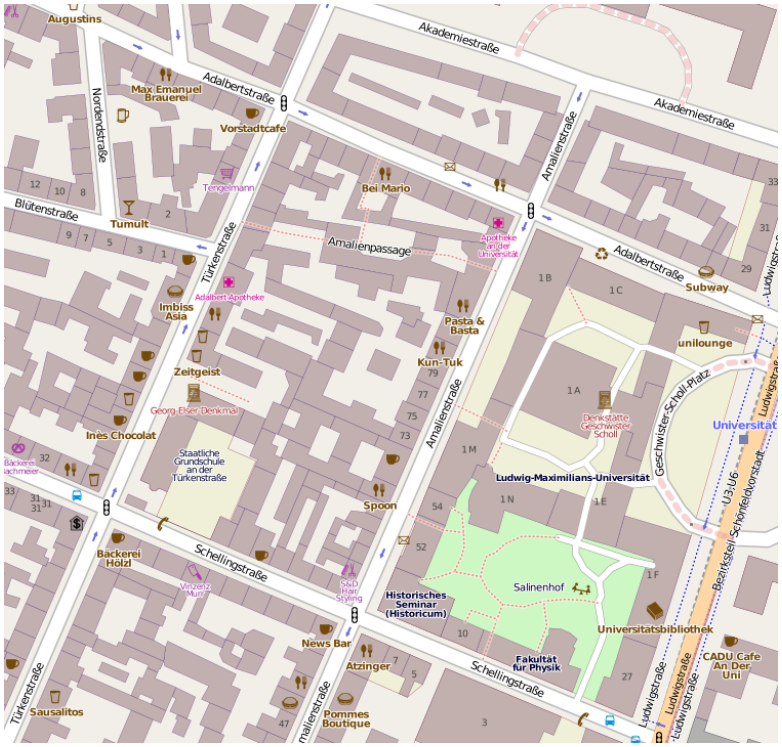
Mit einem Klick auf Daten gelangt man in die Datenansicht (Abb.4, S.13). Es werden nun alle sichtbaren Datenobjekte (Kapitel 3.1) geladen (kann unter Umständen etwas länger dauern) und angezeigt. Mit Klick auf die jeweiligen roten bzw. grünen Objekte sieht man links die jeweilige Beschreibung im Detail.

Abbildung 2: Zusätzliche Optionen beim Drücken des +-Symbols.

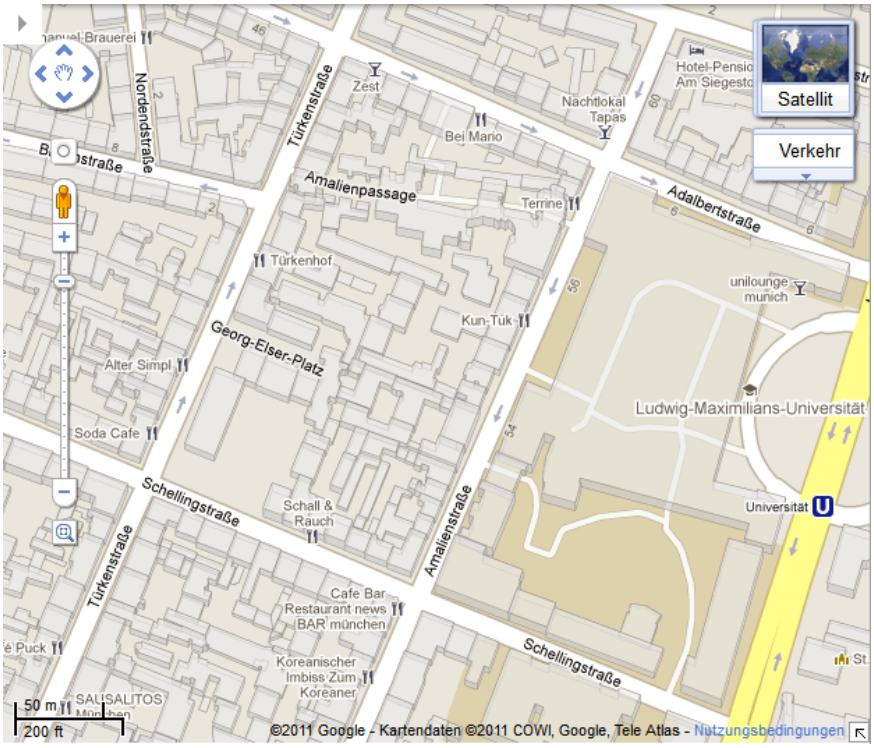
Am oberen Bildrand befinden sich noch weitere Buttons zum Bearbeiten der Daten mit dem Editor *Potlatch 2* (Näheres in Ramm and Topf (2010, Kapitel 10)) oder zum Export der Daten in unterschiedlichen Formaten (XML, HTML, JPG, PNG). Die Chronik-Funktion gibt die letzten Änderungen im ausgewählten Gebiet an. Allerdings nicht so schnell verständlich wie bei *Wikipedia*. Unter GPS-Tracks gibt es die Möglichkeit eigene GPS-Routen hinzuzufügen oder sich welche anzuschauen.

Neben der Haupt-*OpenStreetMap*-Webpage gibt es noch viele weitere Projekte, die auf den Daten von *OpenStreetMap* basieren, beispielsweise:

- Routenplanung unter `www.openrouteservice.org`: Hier ist vor allem die Funktion für Fahrräder und Fußgänger interessant, da *OpenStreetMap* nicht grundsätzlich auf Autos ausgelegt ist, wie andere Vertreter von Onlinekartendiensten.
- `http://www.wanderreitkarte.de/`: Eine Karte, die sich auf Wander- und Reitwege spezialisiert hat.
- `http://wiki.openstreetmap.org/wiki/List_of_OSM_based_Services`: Hier findet man noch weiter Projekte, die auf den Daten von *OpenStreetMap* basieren.



(a) *OpenStreetMap* © *OpenStreetMap* & Contrib, CC-BY-SA



(b) *GoogleMaps* ©2011 Google - Kartendaten ©2011 COWI, Google, Tele Atlas

Abbildung 3: Kartenausschnitt aus *OpenStreetMap* (a) und *GoogleMaps* (b)

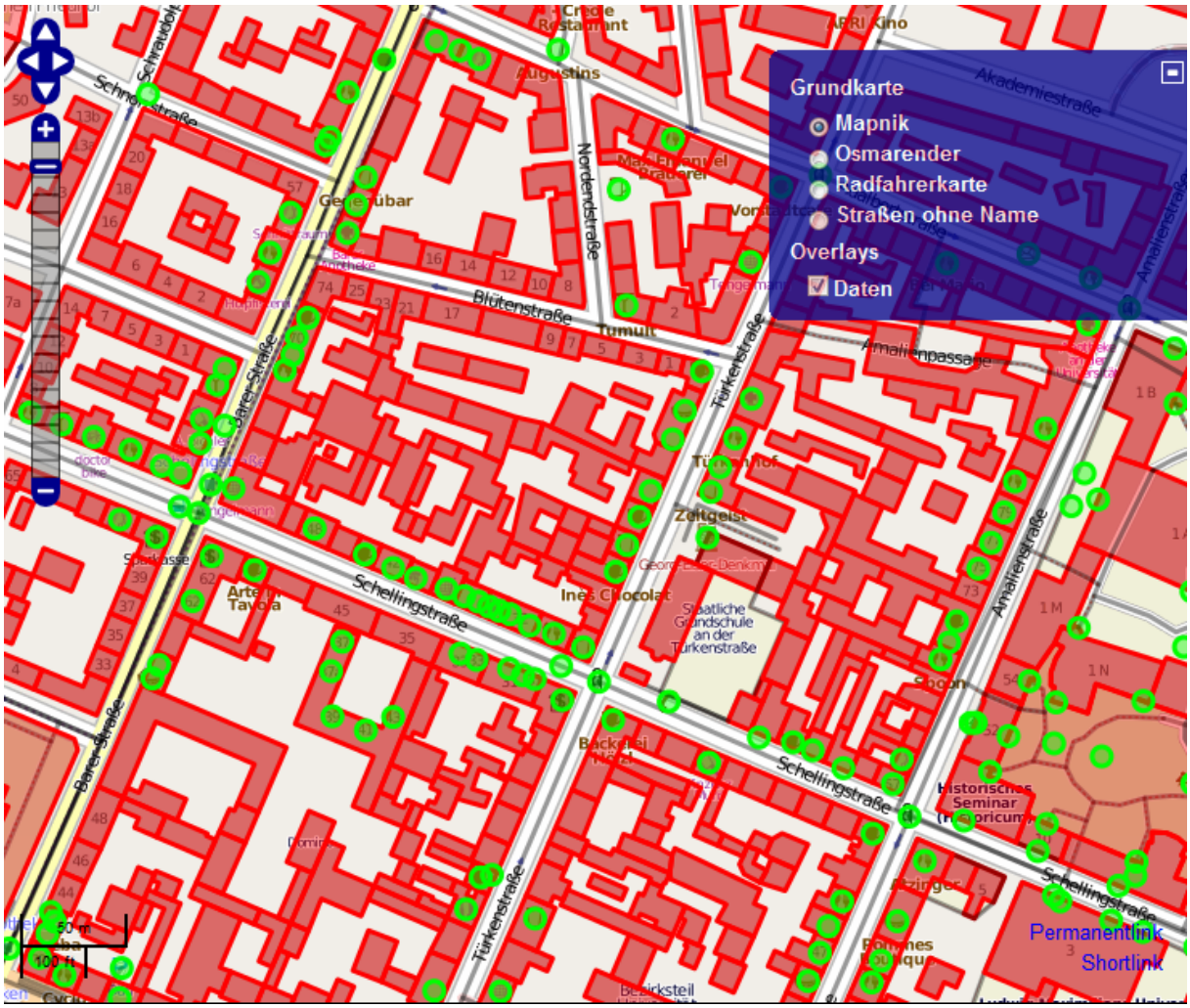


Abbildung 4: Datenansicht bei *OpenStreetMap*.

2.3 Lizenz

Der Begriff *frei* in *freie Daten* ist nicht gleich zu setzen mit *kostenlos*, denn frei wird im Sinne der *Open Source*-Bewegung genutzt und kostenlose Onlinekartendienste gibt es viele. Der Unterschied liegt in der Nutzung der verfügbaren Karten. Bei anderen Vertretern, z.B. *GoogleMaps*, kann es teilweise schon zu Problemen kommen, wenn eine einfache Anfahrtsskizze in einer Broschüre benutzt wird. Denn die geographischen Daten unterliegen den Rechten des jeweiligen Anbieters, die oft eingeschränkt sind. (Ramm and Topf, 2010, S.4f)

Die Lizenz unter der die Daten in *OpenStreetMap* stehen ist die **Creative Commons Attribution-ShareAlike 2.0**[1](kurz: CC-BY-SA 2.0). Man darf die Daten somit kopieren, weiterverarbeiten und veröffentlichen. Allerdings nur unter der Bedingung es auch unter der *CC-BY-SA 2.0*-Lizenz zu machen. Einmal unter dem Attribut *frei* ausgezeichnete Daten sollen nicht nachträglich Nutzungseinschränkungen unterworfen werden. (Ramm and Topf, 2010, S.241)

Zusätzlich ist der Rechteinhaber der Daten zu nennen. Da dies bei einem Community-Projekt problematisch sein kann, weil die Daten aus vielen unterschiedlichen Quellen stammen, hat sich durchgesetzt „OpenStreetMap (and) contributors“ anzugeben. Das nennen jedes einzelnen Nutzers wäre zu aufwendig und würde dem Sinn des Projektes auch entgegenwirken. Zusammenfassend kann man sagen, dass sämtliche verwendete Daten unter der *CC-BY-SA 2.0*-Lizenz veröffentlicht und *OpenStreetMap* genannt werden muss. In welchem Maße die Kennzeichnung erfolgen muss ist von dem veröffentlichten Werk abhängig. (OpenStreetMap, 2011b)

In einem Beispiel werden auch Vor- und Nachzüge *freier Daten* klar. Man könnte einen Atlas entwerfen, der auf Daten und Karten von *OpenStreetMap* basiert. Diesen Atlas kann man problemlos unter den oben genannten Bedingungen veröffentlichen und verkaufen und damit sein Geld verdienen. Allerdings wäre es nun kein Problem, wenn eine andere Person diesen Atlas nimmt, ihn kopiert und um den halben Preis verkauft. Da der Atlas unter der *CC-BY-SA 2.0*-Lizenz gedruckt wurde, ist dies erlaubt.

Seit 2007 gibt es eine weitreichende Diskussion darüber, ob von der bisherigen *CC-BY-SA 2.0*-Lizenz zur **Open Database Licence**[2](kurz: ODbL) gewechselt werden soll. Gründe hierfür sind zum Beispiel die eigentlich verpflichtende Nennung jedes Mitwirkenden und nicht nur *OpenStreetMap*s. Außerdem gibt es unterschiedliche Auffassungen des Urheberrechts in verschiedenen Staaten der Erde. Denn in vielen Ländern gilt der Grundsatz „Facts are free“ (somit auch *OpenStreetMap*-Daten) und das Urheberrecht (also *Creative Commons*-Lizenzen) gilt nur auf künstlerische Werke.

Weiteres zur Problematik findet man im OSMblog (Wochennotizteam, 2011) oder Ramm and Topf (2010, S.245ff). Aktuelles findet man immer im *OpenStreetMap*-Wiki unter `http://wiki.openstreetmap.org/wiki/Main_Page`. Nach Stand der Bachelorarbeit ist es allerdings so, dass heruntergeladene Daten unter der *CC-BY-SA 2.0*-Lizenz

[1] `http://creativecommons.org/licenses/by-sa/2.0/`
[2] `http://opendatacommons.org/licenses/odbl/`

verbreitet werden müssen, also solche mit denen im osmar-Paket in **R** gearbeitet wird.

2.4 Herkunft der Daten

OpenStreetMap ist eine große Datenbank, von der man Daten hoch- bzw. herunterladen kann. Da man in der Statistik auch an der Quelle und Qualität der vorhandenen Daten interessiert ist, wird nun erläutert, wo die meisten Daten herkommen. Es gibt im üblichen drei Wege der Datenherkunft.

Der erste Weg ist der klassische Weg. Personen nehmen sich ein GPS-Gerät und laufen oder fahren Straßenzüge und Orte ab. Dabei ist es wichtig Wegpunkte in der Umgebung (Kreuzungen, Straßennamen, usw...) aufzuzeichnen. Anschließend lädt man die GPS-Daten auf den *OpenStreetMap*-Server hoch und erweitert sie mit den aufgezeichneten Daten. Ein GPS-Gerät ist auf ± 5 Meter genau, aber bei Land- und Stadtkarten ist das kein Problem. Da viele Wege mehrmals abgelaufen werden, kann man seine Daten mit denen von anderen Nutzern abgleichen und „mitteln“.(Ramm and Topf, 2010, S.30)

Der zweite Weg besteht darin Satellitenbilder abzuzeichnen. Die beiden wichtigsten Quellen hierfür sind die Luftbilder von *Yahoo* und die Landsat-Bilder der NASA. Von Vorteil ist das Abzeichnen von Luftbildern vor allem für Verläufe von Stadt- und Waldgrenzen oder Flussverläufen. Bei *Yahoo* muss man allerdings bezüglich der Rechte aufpassen, denn Informationen über die Straße (Straßennamen, usw...) dürfen nicht kopiert werden. Lediglich Information, die durch die Luftbilder ersichtlich sind. (Ramm and Topf, 2010, S.51f)

Der dritte Weg besteht durch Datenimporte von offizieller Seite. Diese müssen jedoch mit dem „Datenspender“ abgesprochen sein, damit es nicht zu rechtlichen Problemen kommt. Größere Import-Projekte, die schon abgeschlossen sind: Daten aus der *CIA World Database*, das Straßennetz der Niederlanden und Küstenlinien der US-Behörde NGA.(Ramm and Topf, 2010, S.304ff)

Im Allgemeinen gilt allerdings der Grundsatz jeder Community-Datenbank, dass die Daten nur so gut sind, wie die Benutzer, der sie hochgeladen hat. Allerdings werden falsche Daten schnell entdeckt und es sind passende Projekte vorhanden um Bugs in *OpenStreetMap* auszumerzen.

3 Datenmodelle

Dieses Kapitel beschreibt folgende Datenmodelle: (1) das *OpenStreetMap*-Datenmodell und (2) das sp-Datenmodell. Der dritte Teil erklärt die Konvertierung von (1) in (2) und führt neue Strukturen für nicht passende Daten ein.

3.1 Datenmodell in *OpenStreetMap*

Die in *OpenStreetMap* beschriebenen Informationen sind in Form einer spezifischen Datenstruktur gespeichert. In der aktuellen Version (API 0.6 (OpenStreetMap, 2009)) besteht diese aus drei Objekten: den Nodes (Kapitel 3.1.2), den Ways (Kapitel 3.1.3) und den Relations (Kapitel 3.1.4). Es gibt zwischen diesen Gemeinsamkeiten und auch spezifische Unterschiede. Da das XML-Format die Basis ist, in denen *OpenStreetMap*-Daten ausgetauscht werden, wird dies für die anschaulichere Darstellung verwendet. Eine vereinfachte grafische Darstellung folgt in Abbildung 5 (S.21).

Die folgenden Ausführungen basieren zum Großteil auf OpenStreetMap (2011a) und Ramm and Topf (2010, Kapitel 6).

3.1.1 Grundlegende Objektstruktur

Prinzipiell handelt es sich bei *OpenStreetMap*-Objekten um Objekte, die eine geographische Information speichern und über sogenannte „Tags“ eine Bedeutung erhalten. Die Basisstruktur der *OpenStreetMap*-Datenbank sieht im XML-Format folgendermaßen aus:

```
<osm version="0.6" generator="OpenStreetMap Server">
   <Objekt1 [Attribute]>
       <Tag1/>
       <Tag2/>
   </Objekt1>
   <Objekt2 [Attribute]>
       <Tag1/>
   </Objekt2>
</osm>
```

Ein Objekt kann entweder Node, Way oder Relation sein und bestimmte Attribute besitzen. Einige Attribute sind bei jedem Objekttyp gleich, hier mit erfundenem Inhalt:

```
<Objekt id="12345" visible="true" timestamp="2005-01-01T23:45:25Z"
   version="2" changeset="54321" user="JohnDoe" uid="9876">
```

`Objekt` ist im Beispiel ein Platzhalter für einen der drei Objekttypen. Jedes Objekt erhält eine innerhalb des Objekttyps eindeutige `id`. Das bedeutet die ID `"12345"` kann sowohl für ein Node, Way oder Relation stehen. Eine einmal vergebene ID ist auch eindeutig in dem Sinne, dass sie nicht mehr von einem anderen Objekt verwendet werden darf. Diese Regel bleibt auch im Falle einer Löschung, denn die ID bleibt dem Objekt zugeordnet. Im Falle eines gelöschten Objektes würde bei `visible="false"` stehen.

Unter `timestamp` und `version` befinden sich das letzte Änderungsdatum im *W3C Date and Time format* (`www.w3.org/TR/NOTE-datetime`) und die Version, in der sich das Objekt befindet. Die Versionsnummern starten bei 1, d.h. das Beispielobjekt wurde bisher einmal verändert. Zum Nachvollziehen der Änderungen ist die Nummer `"54321"` unter `changeset` angegeben. Dies beschreibt die Nummer eines Changeset-Objektes, in dem Änderungen zum Beispielobjekt von Version 1 auf Version 2 stehen. Der Verantwortliche der letzten Änderung ist mit einer eindeutig zugeordneten user ID (`uid`) und einem Usernamen (`user`) identifiziert. Es ist dem User erlaubt den Usernamen im Laufe seiner *OpenStreetMap*-Aktivitäten zu ändern, aber die user ID bleibt ihm erhalten.

Unter `<Tag1/>` sind Daten, die das Objekt genauer beschreiben, enthalten. Ein Tag hat immer die Struktur `key=value`, z.B. `name="Leopoldstraße"`. Schlüssel und Werte sind in UTF-8 kodiert und dürfen je 255 (prinzipiell) beliebig wählbare Zeichen haben, wobei nur Werte eine Zeichenlänge von Null haben dürfen. Seit der API 0.6 gilt zusätzlich die Restriktion, dass Schlüssel nur einmal pro Objekt vorkommen dürfen.

Damit man genug Freiheit besitzt, das Objekt außreichend zu beschreiben, gibt es keine Beschränkung bei der Anzahl der Tags, die ein Objekt enthalten darf. Die Art, wie die Schlüssel formuliert werden dürfen ist ebenfalls frei wählbar. Mittlerweile haben sich allerdings Schlüssel eingebürgert, die häufig bei der Beschreibung bestimmter Objekte verwendet werden. Hierzu mehr in den Abschnitten 3.1.2 - 3.1.4.

3.1.2 Node

Das Basiselement des *OpenStreetMap*-Datenmodells sind die Nodes (*engl.: Knoten*). In XML-Darstellung sieht ein Brunnen beispielsweise so aus (Originaldaten):

```
<node id="201978860" lat="48.1506744" lon="11.5809644"
  changeset="8397338" user="KLMB" uid="342705" visible="true"
  timestamp="2011-06-10T13:34:48Z" version="4">
   <tag k="lit" v="yes"/>
   <tag k="amenity" v="fountain"/>
   <tag k="drinkable" v="no"/>
</node>
```

Zusätzlich zu den in Kapitel 3.1.1 erläuterten Attributen (ID, User- & Änderungsinformationen) sind nun geographische Angaben unter `lat` und `lon` enthalten. Jeweils auf bis zu sieben Nachkommastellen enthalten die Attribute die geographische Breite im Bereich von -90 bis $+90$ Grad und die geographische Länge im Bereich -180 bis $+180$ Grad. Basierend auf dem WSG84-Referenzsystem entspricht das einem Genauigkeitsgrad von ± 1 cm.

In den drei Tags ist nun erkennbar, wie die beschriebene `key=value`-Kombination (abgekürzt mit `k="... v="..."`) aussieht. Dies ist ein Brunnen (`amenity`), der beleuchtet ist (`lit`) und dessen Wasser nicht trinkbar ist (`drinkable`).

Grundsätzlich gibt es zwei Arten von Nodes. Entweder sie haben keine Daten (Tags) und bestehen nur aus Attributen. Dies ist vor allem der Fall, wenn sie zur Bildung von

Ways benutzt werden (siehe Kapitel 3.1.3). Der andere Fall ist, sie beschreiben *Points of Interest* (z.B. Ampeln, Restaurants, Briefkästen, ...). Die am häufigsten benutzten Keys sind `amenity` (*engl*: Einrichtung) und `name` (Schlüssel-Information aus Topf, 2010).

3.1.3 Way

Das zweite Grundelement im *OpenStreetMap*-Datenmodell sind die Ways. Üblicherweise werden sie zur Darstellung linienförmiger Objekte benutzt. Beispiele hierfür wären Straßen, Flussverläufe oder ähnliches. Als Beispiel folgt eine kleine Straße im XML-Format (Originaldaten):

```
<way id="73207194" visible="true" timestamp="2011-01-22T15:00:41Z"
  version="4" changeset="7051198" user="KLMB" uid="342705">
   <nd ref="18280282"/>
   <nd ref="1115463903"/>
   <nd ref="1078808643"/>
   <nd ref="868440268"/>
   <nd ref="18280280"/>
   <tag k="highway" v="residential"/>
   <tag k="lanes" v="1"/>
   <tag k="oneway" v="yes"/>
</way>
```

Die Grundattribute (Kapitel 3.1.1) sind enthalten, aber im Vergleich zur Node (Kapitel 3.1.2) stehen die geographischen Informationen nicht direkt im Kopf des Way-Objektes sondern in Tags der Art `<nd ref="<Id>"/>`. Dies sind die IDs der Nodes, die den Weg definieren (im Beispiel ist der Way durch fünf Nodes definiert). Die Grenzen eines Ways liegen bei mindestens 2 Nodes und höchstens 2.000 Nodes und werden in der Reihenfolge ausgegeben, in der die Linie verläuft. Einer der häufigsten Schlüssel für Linien ist `highway`, denn er bestimmt die Verkehrsbedeutung der Straße (Fahrradweg, Gleise, Stadtstraße, ...) (Schlüssel-Information aus Topf, 2010).

Im folgenden Beispiel wird ein Way dargestellt, der eine Fläche definiert:

```
<way id="80175505" visible="true" timestamp="2010-10-02T16:23:59Z"
  version="1" changeset="5937536" user="KLMB" uid="342705">
   <nd ref="935274662"/>
   <nd ref="935275148"/>
   <nd ref="935275414"/>
   <nd ref="935274970"/>
   <nd ref="935274662"/>
   <tag k="building" v="yes"/>
</way>
```

Der Datenbank-Server erkennt eine Fläche daran, dass die erste und letzte Noderefernz dieselbe ist und passende Schlüssel verwendet wurden. Zu diesen Schlüsseln gehören (wie im Beispiel) `building` oder `area="yes"`. Nur dieselbe Anfangs- und Endnode rei-

chen nicht, denn es existieren auch geschlossene Linien (beispielsweise ein Kreisverkehr). In früheren Versionen der Datenbank gab es auch eigene Area-Objekte, die sich allerdings nicht durchsetzen konnten. Die Modellierung von Flächen findet nun mit Hilfe von Ways oder Relations (Kapitel 3.1.4) statt.

3.1.4 Relation

Relations sind das dritte Grundelement im *OpenStreetMap*-Datenmodell. Ihr Zweck ist es, Beziehungen zwischen verschiedenen Objekten (Nodes, Ways oder anderen Relations) darzustellen. Typische Beispiele wären das Zusammenfügen mehrerer Ways und Nodes zu einer Route oder einer komplexen Fläche. Als Beispiel folgt eine Abbiegevorschrift (Originaldaten):

```
<relation id="373395" visible="true" timestamp="2010-01-13T19:35:29Z"
  version="5" changeset="3612328" user="JosmJo" uid="118197">
   <member type="way" ref="48242678" role="to"/>
   <member type="node" ref="27270924" role="via"/>
   <member type="way" ref="47229081" role="from"/>
   <tag k="restriction" v="only_right_turn"/>
   <tag k="type" v="restriction"/>
</relation>
```

Neben den Attributen sind die Objekte zwischen denen eine Beziehung herrscht unter `member` aufgelistet. `type` gibt die Art des Objektes und `ref` die ID des Objektes an. Neu kommt das `role`-Attribut hinzu. Dies nennt welche Rolle das Objekt in der Relation spielt, aber kann ein beliebiger (auch leerer) Freitext sein. Die Tags für die Beschreibung der Daten, sind wie bei den Ways und Nodes (Kapitel 3.1.2 & 3.1.3) im `k="..."v="..."`-Format angegeben. Im oberen Beispiel ist somit angegeben, dass von einem Way zum anderen Way über die Node die Abbiegevorschrift herrscht, nur Rechtsabbiegen zu können.

Für eine Relation braucht man mindestens einen Teilnehmer oder einen Daten-Tag. Bezüglich der `member`-Tags gibt es die Restriktion, dass Objekte beliebig oft vorkommen, aber nicht mit derselben Rolle versehen sein dürfen. Außerdem sind die Teilnehmer geordnet. Ein Schlüssel, der bei fast 100% aller Relations vorkommt ist `type`. Dieser gibt an, was für einen Typ von Relation dargestellt wird (Schlüssel-Information aus Topf, 2010).

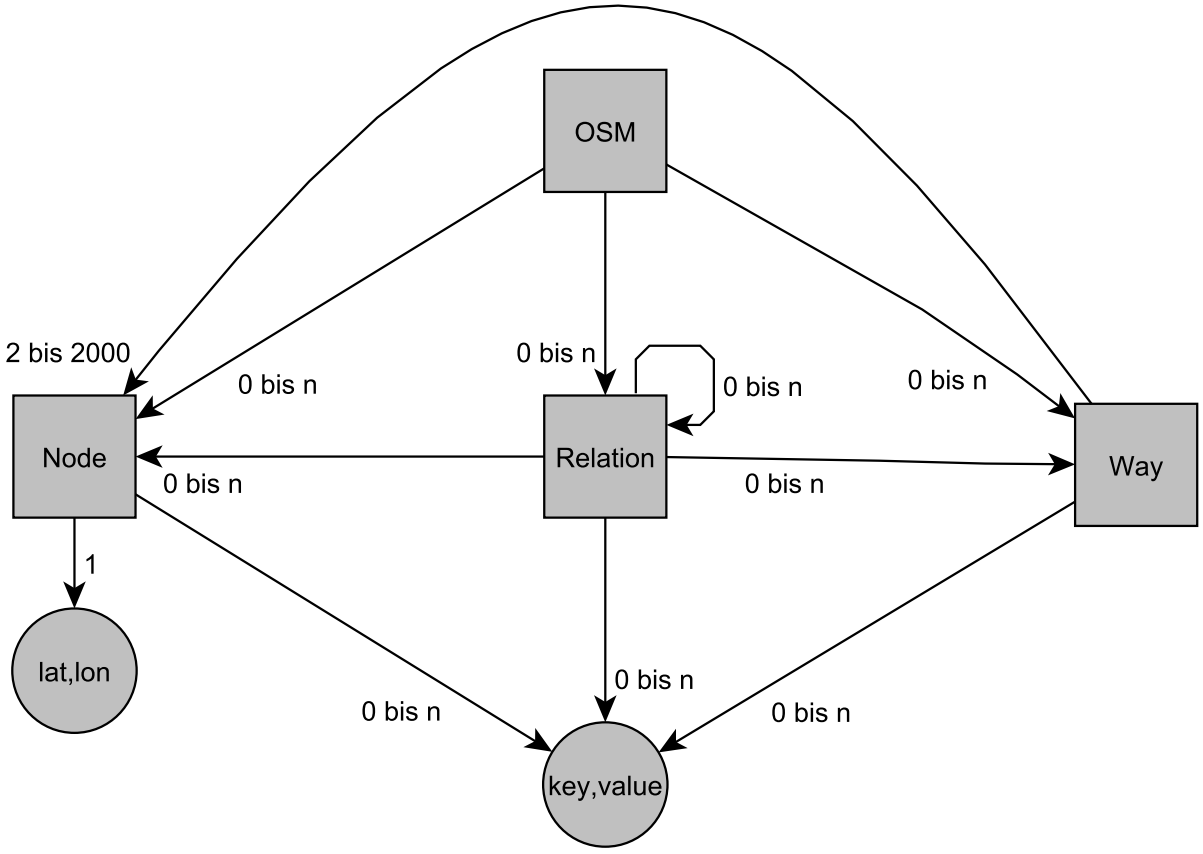


Abbildung 5: vereinfachte Darstellung des *OpenStreetMap*-Datenmodells. *0 bis n* bedeutet Element darf 0 bis n mal vorkommen.

3.2 Datenmodell im Paket sp

Die folgenden Ausführungen basieren zum Großteil auf (Bivand et al., 2008, Kapitel 1-3).

3.2.1 Grundlegendes zu sp

Das Gebiet der räumlichen Datenanalyse ist in der Statistik ein wichtiges Thema. Aus diesem Grund gab es für das Statistikprogramm **R** eine Vielzahl an Paketen, die aus der GIS-Welt (geographical information systems) Strukturen implementiert haben. Die Organisation der Daten war in jedem Paket meistens anwendungsbezogen und unterschiedlich. Eine feste Grundlage, die das Zusammenarbeiten vereinfacht hätte, hat gefehlt. Aus diesem Grund wurde im Jahr 2004 das sp-Paket (Pebesma and Bivand, 2011) veröffentlicht.

Das sp-Paket gibt eine feste Struktur der Datenorganisation räumlicher Daten vor. Die Idee dahinter ist, dass aufbauend auf einer gemeinsamen Grundlage das Arbeiten mit räumlichen Daten vereinheitlicht und gemeinsam weiterentwickelt werden kann. Ein Ziel hierbei ist, dass neben der rein räumlichen Frage *„Wo ist etwas?“*, auch die Frage *„Was ist wo?“* beantwortet werden kann.

Im Allgemeinen wird jede Art von räumlichen Daten in der Basis durch zwei Informationen gekennzeichnet. Die erste ist der geographische Bereich, in dem sich die Daten befinden. Im einfachsten Fall wird dieser durch einen rechteckigen Bereich abgegrenzt. Die zweite Information deckt ab, auf welcher Darstellung der Erdoberfläche man sich befindet. Damit ist die Definition eines *CRS* (*Coordinate Reference System*) gemeint. Ein CRS definiert mit welchem Referenzsystem und welcher Projektion der Erde man arbeitet. Weitere Informationen bzgl Koordinaten-Referenz-Systemen und Projektionen findet man auf Knippers (2009).

Innerhalb der zugrundeliegenden Basis wird mit verschiedenen Arten von Objekten gearbeitet. sp unterscheidet vier Typen:

Point Ein Punkt hat eine Koordinate, die diesen explizit definiert. (siehe Abschnitt 3.2.2)

Line Eine Anzahl an Punkten wird in eindeutiger Reihenfolge miteinander verbunden und definiert eine Linie. (siehe Abschnitt 3.2.3)

Polygon Eine Fläche, die durch eine Anzahl an verbundenen Linien abgegrenzt ist.

Grid Ein Raster mit gleichgroßen rechteckigen Zellen, welches über eine Karte gelegt wird.

Point und Line werden noch detaillierter beschrieben.

Die Implementierung in sp funktioniert über eine S4-Klassenstruktur. Diese erlaubt Objekte mit klar definierten Inhalten und stellt Methoden zur Verfügung diese auf ihre Korrektheit zu überprüfen. Weitere Informationen in Chambers (2008). Eine vereinfachte grafische Darstellung des Datenmodells befindet sich in Abbildung 6 (S.27).

Als Grundlage nimmt sp die Klasse `"Spatial"`. Auf dieser bauen sämtliche anderen Objekttypen auf. Sie beinhaltet die erwähnten Basisinformationen in zwei Slots. Der

als Rechteck definierte Datenbereich ist im Slot `bbox` (BoundingBox) als 2x2-Matrix gespeichert. Im Slot `proj4string` befindet sich ein `CRS`-Objekt, das durch einen String im PROJ.4-Format (weitere Informationen in Evenden and Warmerdam (2000)) kennzeichnet, welches Koordinatenreferenzsystem verwendet wird.

Nun folgt beispielhaft ein in **R** erstelltes `"Spatial"`-Objekt. Es deckt einen rechteckigen Bereich um Deutschland mit dem WGS84-Referenzellipsoiden ab. Zum Umgang mit dem PROJ.4-Format empfiehlt es sich das rgdal-Paket (Keitt et al., 2011) zu laden.

```
> library(sp)
> library(rgdal)
> bereich < -matrix(c(5,48,15,55),ncol=2, dimnames=list(c("lon","lat"),
+            c("min","max")))
       min   max
  lon FALSE FALSE
  lat FALSE FALSE
> crs <- CRS("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs")
> Spatial(bbox=bereich, proj4string=crs)
  An object of class "Spatial"
  Slot "bbox":
      min max
  lon   5  15
  lat  48  55

  Slot "proj4string":
  CRS arguments:
   +proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs
  +towgs84=0,0,0
```

3.2.2 Spatial Points

Der erste Typ von räumlichen Objekten sind Punkte. Ein Punkt ist im geographischen Bereich durch zwei Koordinaten (x,y) definiert. Auf der x-Achse verlaufen die Längengrade von 0° bis 360° bzw. von -180° bis 180°, wobei 0° (je nach gewähltem CRS) bei Greenwich liegt. Die Breitengrade verlaufen auf der y-Achse von -90° am Südpol bis +90° am Nordpol.

In `sp` ist die räumliche Definition von Punkten mit der Klasse `SpatialPoints` abgedeckt. Diese erweitert die Klasse `Spatial` mit dem Slot `coords`, welche die einzelnen Koordinaten der Punkte enthält. Im Beispiel wird ein `SpatialPoints`-Objekt mit drei geographischen Angaben gebildet.

```
> townsGeo <- data.frame(lon=c(11.58, 8.68, 9.99), lat=c(48.15, 50.12, 53.56))
> crs <- CRS("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs")
> SPtowns <- SpatialPoints(townsGeo, crs)
> SPtowns
```

```
  SpatialPoints:
        lon   lat
  [1,] 11.58 48.15
  [2,]  8.68 50.12
  [3,]  9.99 53.56
  Coordinate Reference System (CRS) arguments: +proj=longlat
  +ellps=WGS84 +datum=WGS84 +no_defs +towgs84=0,0,0
> bbox(SPtowns)
       min   max
  lon  8.68 11.58
  lat 48.15 53.56
```

`SPtowns` ist nun ein Objekt, welches in einem durch die `bbox` definierten Bereich drei Punkte darstellt. Die `bbox` muss nun nicht mehr per Hand angegeben werden, denn die `SpatialPoints`-Methode bildet sie selbst. In der Struktur des Objektes sind die drei Komponenten von `SPtowns` ersichtlich:

```
Formal class 'SpatialPoints' [package "sp"] with 3 slots
  ..@ coords     : num [1:3, 1:2] 11.58 8.68 9.99 48.15 50.12 ...
  .. ..- attr(*, "dimnames")=List of 2
  ..@ bbox       : num [1:2, 1:2] 8.68 48.15 11.58 53.56
  .. ..- attr(*, "dimnames")=List of 2
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
```

Gegeben dem Fall, dass für jeden Punkt auch eine andere nicht rein räumliche Information vorliegt, stellt `sp` eine erweitere Art der Datenorganisation zur Verfügung. Diese spaltet die Koordinaten von den neuen Daten und speichert beides in unterschiedlichen Slots ab. Verknüpft werden die Komponenten durch die Nummerierung der Zeilen, welche die Bedeutung einer eindeutig definierten ID erhalten.

```
> townsData <- data.frame(name=c("Mu", "Fra", "HH"),
+                         population=c(1.353, 0.680, 1.786))
> townsData
    name population
  1   Mu      1.353
  2  Fra      0.680
  3   HH      1.786
> SPDFtowns <- SpatialPointsDataFrame(townsGeo, townsData, proj4string=crs)
> SPDFtowns
       coordinates name population
  1 (11.58, 48.15)   Mu      1.353
  2  (8.68, 50.12)  Fra      0.680
  3  (9.99, 53.56)   HH      1.786
```

Das `"SpatialPointsDataFrame"` enthält nun neben den geographischen Informationen auch die nicht-geographischen Informationen. Allerdings ist pro geographischen Punkt nur eine Datenzeile reserviert.

3.2.3 Spatial Lines

Der zweite wichtige Objekttyp in räumlichen Daten sind Linien. Für die Darstellung auf einem Plot ist die simple Denkweise ausreichen: Gegeben einer Matrix von geographischen Punkten verbindet man diese bis eine Lücke auftaucht und zeichnet weiter ab dem Punkt, wo die Lücke aufhört. Für die Verknüpfung mit nicht-geographischen Daten ist dies allerdings nicht genügend.

Das `sp`-Paket bietet eine andere Lösung. Mehrere Linien ohne Lücken (bzw. `NA`'s) bestehend aus 2D-Koordinaten werden in seperaten `Line`-Objekten gespeichert. Ein Zusammenschluss mehrerer solcher Objekte (beispielsweise ein Straßennetz, welches nicht durch eine Linie beschrieben werden kann) wird mit einer eindeutigen ID versehen und als `Lines`-Objekt gespeichert. Durch die ID ist der Zusammenschluss der Einzellinien eindeutig identifizierbar.

Im folgenden Beispiel werden zwei Linien (grob Nord- und Südseite des Bodensees) miteinander zu einem Objekt verbunden. Das Ergebnis zeigt ein `Lines`-Objekt, welches die ID `"1"` besitzt.

```
> bodSouth <- data.frame(lat=c(8.94, 9.49, 9.72), lon=c(47.73, 47.48, 47.50))
> bodNorth <- data.frame(lat=c(9.05, 9.48, 9.75), lon=c(47.8, 47.65, 47.51))
> bodSouthLine <- Line(bodSouth)
> bodNorthLine <- Line(bodNorth)
> bodSouthLine
  An object of class "Line"
  Slot "coords":
        lat   lon
  [1,] 8.94 47.73
  [2,] 9.49 47.48
  [3,] 9.72 47.50
> bodLines <- Lines(list(bodSouthLine, bodNorthLine), ID="1")
> str(bodLines, max.level=3)
  Formal class 'Lines' [package "sp"] with 2 slots
    ..@ Lines:List of 2
    .. ..$ :Formal class 'Line' [package "sp"] with 1 slots
    .. ..$ :Formal class 'Line' [package "sp"] with 1 slots
    ..@ ID   : chr "1"
```

Das `Lines`-Objekt kann allerdings mit willkürlichen Koordinaten, welche nicht den geographischen Längen- und Breitengraden entsprechen, gefüllt werden. Um die Korrektheit zu gewährleisten, können mehrere `Lines` mit eindeutiger ID mit einem `Spatial`-Objekt verknüpft werden. Das Ergebnis ist ein Objekt der Klasse `"SpatialLines"`. Nichtgeographische Informationen werden ähnlich den `SpatialPoints` (Abschnitt 3.2.2) mit Hilfe eines `data`-Slots hinzugefügt, wobei jeder zusammengehörenden Linienkombination eine Datenzeile zur Verfügung steht.

```
> crs <- CRS("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs")
> SPBoden <- SpatialLines(list(bodLines), proj4string=crs)
> str(SPBoden, max.level=2)
  Formal class 'SpatialLines' [package "sp"] with 3 slots
    ..@ lines      :List of 1
    ..@ bbox       : num [1:2, 1:2] 8.94 47.48 9.75 47.8
    .. ..- attr(*, "dimnames")=List of 2
    ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots


> bodenData <- data.frame(deep=254, height=395)
> SPDFBoden <- SpatialLinesDataFrame(SPBoden, bodenData, match.ID=TRUE)
> str(SPDFBoden, max.level=2)
  Formal class 'SpatialLinesDataFrame' [package "sp"] with 4 slots
    ..@ data       :'data.frame':        1 obs. of  2 variables:
    ..@ lines      :List of 1
    ..@ bbox       : num [1:2, 1:2] 8.94 47.48 9.75 47.8
    .. ..- attr(*, "dimnames")=List of 2
    ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
```

Das Ergebnis ist ein `"SpatialLinesDataFrame"`.

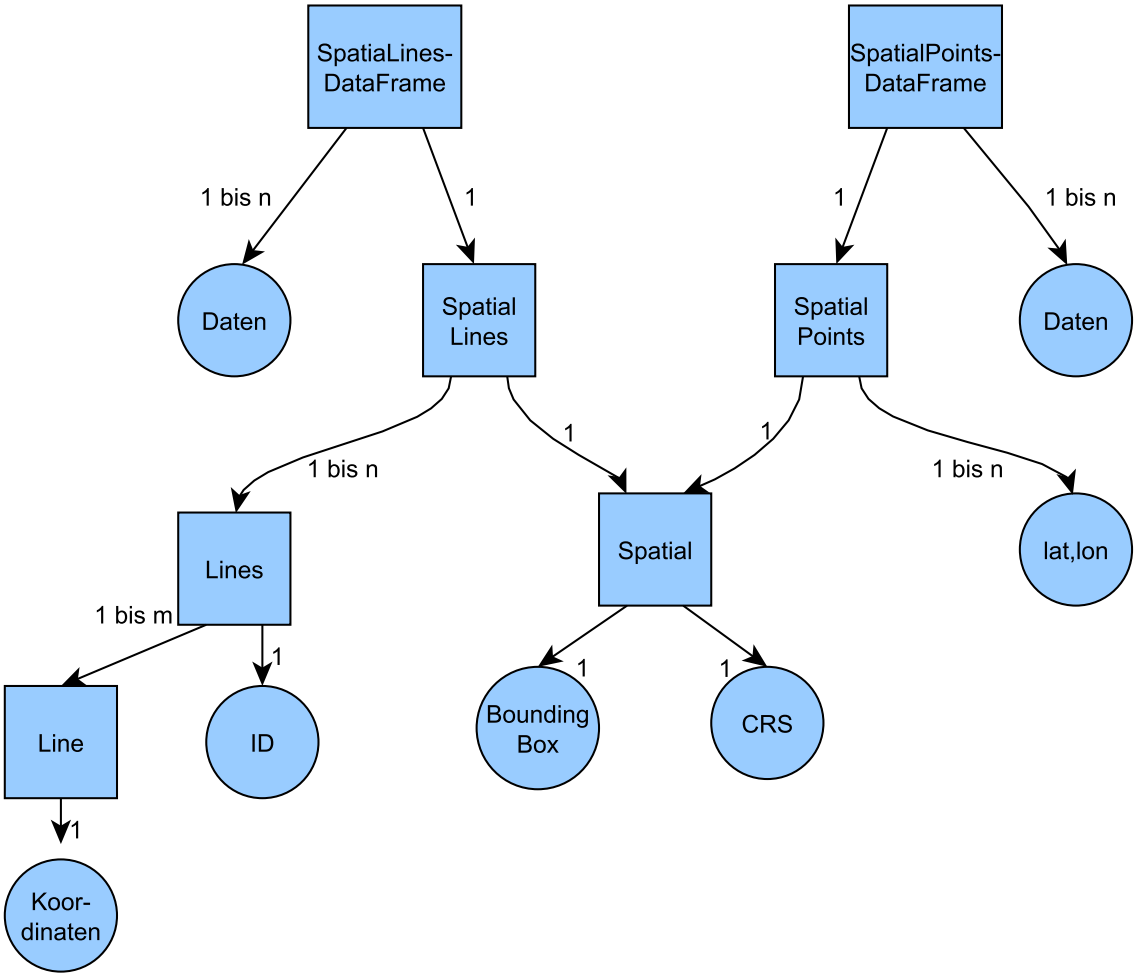


Abbildung 6: vereinfachte Darstellung des sp-Datenmodells. *0 bis n* bedeutet Element darf 0 bis n mal vorkommen.

3.3 Verknüpfung von sp und *OpenStreetMap*

Das in Abschnitt 3.1 beschriebene Datenmodell von *OpenStreetMap* lässt sich neben der Aufteilung in Node, Way und Relation noch in geographische und nicht-geographische Informationen einteilen. Geographische Daten werden soweit es möglich ist in `sp`-Objekte konvertiert und nicht-geographisches wird in eigene Strukturen verpackt.

Wichtig war mir, dass die Konvertierung vorerst unabhängig von den Tags (`key=value`) möglich ist. Denn durch die endlosen Kombinationsmöglichkeiten und teilweisen Inkonsistenz, lassen sich Automatisierungen schwer realisieren.

3.3.1 Geo-Daten

Von den drei *OpenStreetMap*-Objekttypen enthalten nur Nodes und Ways immer geographische Informationen in Form von Positionen auf dem Koordinatensystem. Relations müssen nicht zwingend Geo-Daten enthalten, weshalb sich die Überführung in `sp`-Objekte als problematisch darstellt.

Das Node-Äquivalent in `sp` ist das `SpatialPointsDataFrame` (Kapitel 3.2.2). Da die Koordinaten der Nodes auf dem vom GPS-System verwendeten WGS84-Referenzsystem basieren, wird ein passender `CRS`-String verwendet (`"+proj=longlat +ellps=WGS84 +datum=WGS84"`). Weiterhin werden die Längen- und Breitengrade jedes Nodes mit diesem verknüpft und ein `SpatialPoints`-Objekt konstruiert. Die BoundingBox wird automatisch erstellt.

Die Attribute jedes Node-Objektes enthalten immer dieselben Merkmale mit jeweils einer Merkmalsausprägung. Gespeichert werden die Attribute als `data.frame` mit der ID als row.number im `data`-Slot des `SpatialPointsDataFrame`.

Das nächste Objekt wäre das den Ways ähnelndem `SpatialLinesDataFrame`. Die Koordinaten eines Ways sind nicht explizit aufgeführt, sondern über die referenzierten Node-IDs zu bekommen. Mithilfe der Nodes konstruiert man eine Koordinatenmatrix und somit ein `Line`-Objekt. Dies gibt man direkt mit der ID an ein `Lines`-Objekt weiter. Da die Tags vorerst ignoriert werden, unterscheidet man nicht zwischen geschlossenen (z.B. Flächen, Gebäuden) und nicht-geschlossenen Linien. Eine Koordinate wird im Fall einer Fläche doppelt im `Line`-Objekt gespeichert.

Eine Liste mit `Lines` bildet zusammen mit dem passenden `CRS`-String das `SpatialLines`-Objekt. Als CRS gibt man, wie bei den Nodes, das WGS84-Referenzsystem an. Attribute haben auch bei den Ways immer dieselben Merkmale mit jeweils einer Merkmalsausprägung und werden mit der ID als row.number im `data`-Slot des `SpatialLinesDataFrame` gespeichert.

3.3.2 Nicht-Geo-Daten

Nachdem Geo-Daten nun im `sp`-Format sind, müssen noch die Relations und Tags der Nodes und Ways konvertiert werden. Bei den Tags stellt sich hierbei eine Problematik ein, denn das übliche „wide“-Format mit einer Zeile pro ID funktioniert nicht. Spaltennamen

würden dann Schlüssel enthalten und die Merkmalsausprägungen wären die dazugehörigen Werte. Folgendes Beispiel veranschaulicht dies:

```
          id                     name building        street    amenity
1 1318875603            Eulenbrunnen     <NA>          <NA>       <NA>
2  962182561                    <NA> entrance          <NA>       <NA>
3 1345067162 Institut fuer Statistik     <NA> Ludwigstrasse university
```

Durch das Vorhandensein von unterschiedlichen Schlüsseln in jedem Objekt, hätte das Dataframe eine potentiell unbegrenzte Anzahl an Spalten. Viele wären außerdem mit `NA`'s gefüllt, da der Schlüssel nicht im jeweiligen Objekt vorkommt. Das Dataframe verliert dadurch an Übersichtlichkeit.

Die Lösung des Problems ist das „long"-Format:

```
          id      key                   value
1 1318875603     name             Eulenbrunnen
2  962182561 building                 entrance
3 1345067162     name Institut fuer Statistik
4 1345067162   street            Ludwigstrasse
5 1345067162  amenity               university
```

Hier werden die vorherigen Spaltennamen zu Merkmalsausprägungen eines neuen Merkmals zusammengefasst. Der Vorteil dieser Darstellungsweise ist die Reduzierung der `NA`'s und die Möglichkeit beliebig viele Schlüssel-Wert-Kombinationen mit einer ID zu verknüpfen.

Das „long"-Format wurde bei sämtlichen Tags des Typs `key="value"` verwendet und unter Daten gespeichert. Im Fall von Ways bzw Relations treten noch Tags vom Typ `nd ref="<id>"` und `member type="<type>"ref="<id>"role="<role>"` auf. Diese werden seperat unter Member mit den Spalten `"id"` und `"ref"` bzw. `"type"`, `"ref"` und `"role"` gespeichert.

Bei den Relations fehlt noch die Konvertierung der Attribute, da sie nicht in einem sp-Objekt enthalten sind. Sie werden im „wide"-Format unter Meta gespeichert.

3.3.3 Endgültige Struktur

Nach der Konvertierung der einzelnen Teile wurden diese in ein gemeinsames Konzept gebracht. Die Struktur funktioniert ohne inhaltliche Interpretation der Tags.

Prinzipiell lässt sich zwischen vier Objekttypen unterscheiden. Die geographischen Informationen sind in sp-Objekte konvertiert. Objekte mit Namen „`<typ>DataFrame`" enthalten die nichtgeographischen Daten im long-Format. Innerhalb der Elemente referenzierte Elemente (wie bei Ways und Relations) sind in „`<typ>Member`" ebenfalls im long-Format enthalten. Weiterhin sind in Objekten vom Typ „`<typ>Meta`" die Attribute enthalten. Für `<typ>` ist jeweils `node, way` oder `relation` einzusetzen.

Im Detail sieht die Struktur folgendermaßen aus:

1. `Node`
    - `nodeDataFrame`
    - `SpatialPointsDataFrame`
2. `Way`
    - `wayDataFrame`
    - `SpatialLinesDataFrame` oder `wayMeta`
    - `wayMember`
3. `Relation`
    - `relationMeta`
    - `relationDataFrame`
    - `relationMember`

Im Falle des `Way`-Objektes gibt es allerdings noch einen Spezialfall. Näheres dazu in Abschnitt 4.2.2. Eine Visualisierung der Konvertierung der *OpenStreetMap*-Objekte befinden sich auf Abb. 7 (S.31), Abb. 8 (S.32) und Abb. 9 (S.33).

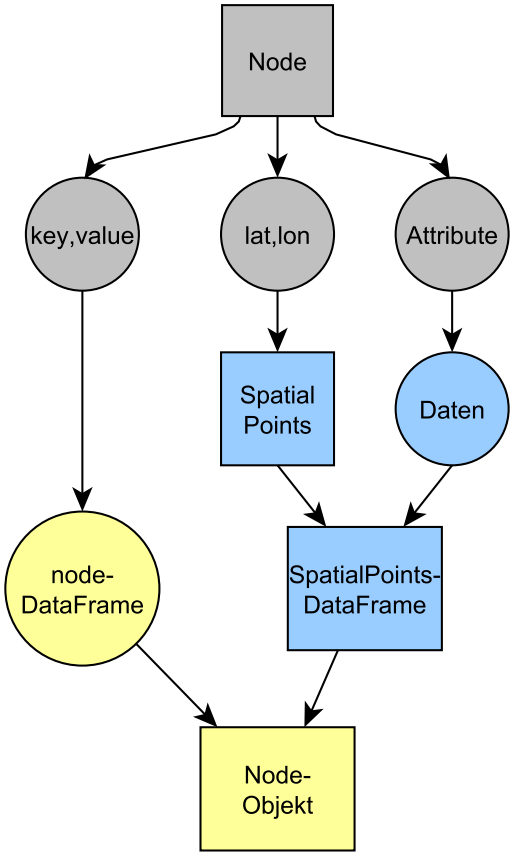


Abbildung 7: Konvertierung von *OpenStreetMap*-Node-Objekten (grau) in sp-Objekte (blau) und osmar-Objekte (gelb)

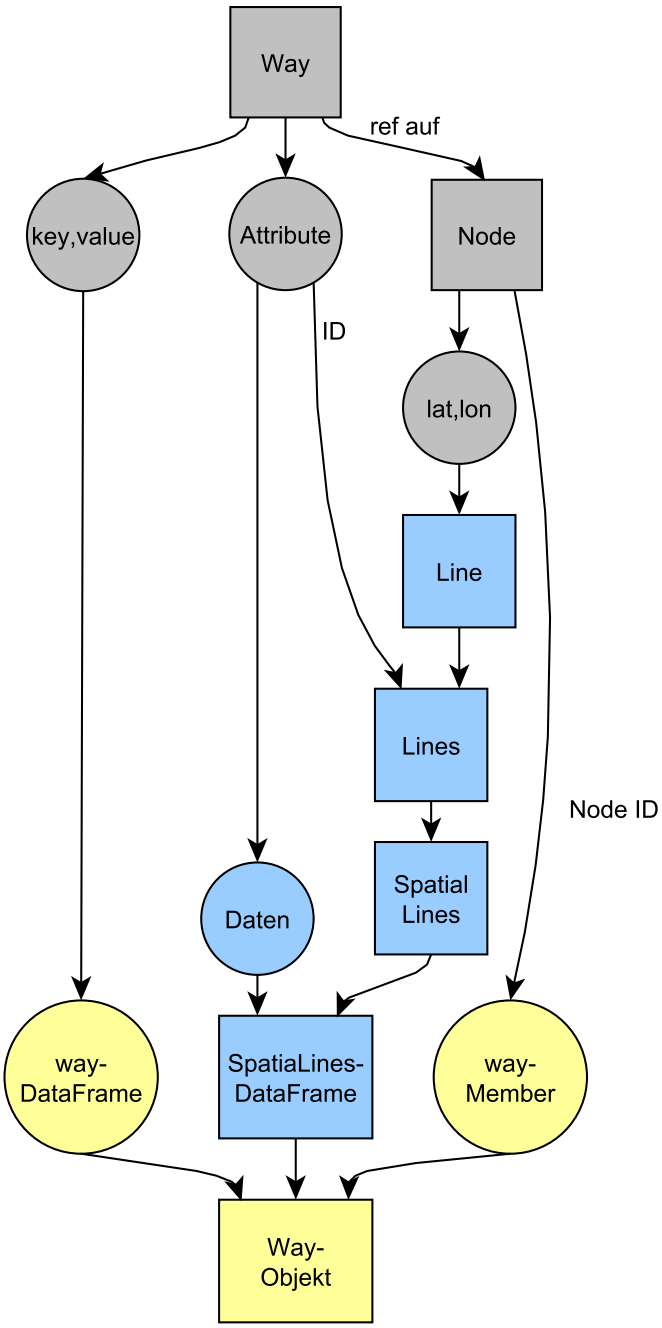


Abbildung 8: Konvertierung von *OpenStreetMap*-Way-Objekten (grau) in sp-Objekte (blau) und osmar-Objekte (gelb)

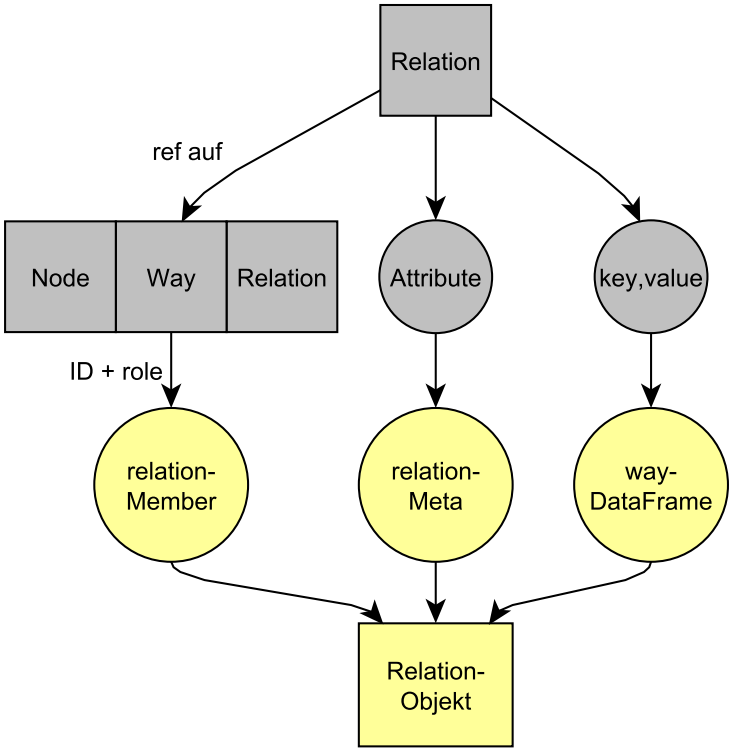


Abbildung 9: Konvertierung von *OpenStreetMap*-Relation-Objekten (grau) in osmar-Objekte (gelb)

4 osmar-package

Dieses Kapitel stellt den Zugriff auf *OpenStreetMap*-Daten und die Implementierung und wichtigsten Funktionen des osmar-Paketes vor.

4.1 *OpenStreetMap*-API

Die folgenden Ausführungen basieren auf der Internetseite der API v0.6 (OpenStreetMap, 2009) und Ramm and Topf (2010, Kapitel 22).

Auf *OpenStreetMap*-Daten kann mit Hilfe eines *Application Programming Interface*'s (API) zugegriffen werden. Es befindet sich seit dem 17.04.2009 in der Version 0.6. Basierend auf dem HTTP-Protokoll findet der Zugriff über die URL „`http://api.openstreetmap.org/api/0.6/...`“ statt. Im Normalfall ist das auch der Weg auf die Daten zuzugreifen, da man nicht mit der Datenbank direkt arbeiten kann.

Das Arbeiten mit der API erfordert die Kenntnis des OSM-XML-Formats (Kapitel 3.1), da sämtliche Daten mit diesem ausgetauscht werden. Im Allgemeinen unterscheidet man drei Operationen:

PUT Ermöglicht die Neuanlage bzw. das Ändern (Hochladen) bestimmter Objekte.

GET Ermöglicht das Lesen (Herunterladen) von Objekte.

DELETE Ermöglicht das Löschen von Objekten.

Im Weiteren sind in osmar vorerst Funktionen vom Typ GET implementiert. Aus diesem Grund wird nur auf diese weiter eingegangen. Prinzipiell wird die URL „`http://api.openstreetmap.org/api/0.6/`“ mit weiteren Pfadangaben erweitert um spezifische Objekte zu erhalten und im Browser eingegeben. Je nachdem finden unterschiedliche Zugriffe statt.

Eine Methode besteht darin einen rechteckigen Bereich (BoundingBox) mit vier Koordinaten festzulegen und Daten herunterzuladen. An die Request-URL muss hierfür „`/map?bbox=<left>,<bottom>,<right>,<top>`“ angehängt werden. Wobei `<...>` einen Platzhalter für die jeweiligen Koordinaten im Längen- und Breitengrad-Format darstellt.

Man bekommt Objekte in einer bestimmten Weise zurück:

1. Sämtliche Nodes im angegebenen Bereich.

2. Alle Ways, die durch den Bereich führen inklusive referenzierter Nodes. Die Nodes müssen allerdings nicht in der BoundingBox vorhanden sein.

3. Alle Relations, die als Mitglied eine der in 1. oder 2. heruntergeladenen Objekte besitzen. Allerdings werden weitere Nodes und Ways der Relations nicht heruntergeladen, da es sonst unendlich weiter gehen könnte.

Zusammenfassend heißt dies, dass man mehr erhält als der Bereich angibt, da aufgrund von „Durchgangsways“ noch Nodes knapp außerhalb des Bereichs heruntergeladen werden.

Das Ergebnis kommt wie in Abschnitt 3.1.1 dargestellt zurück. Zusätzlich gibt es noch Restriktionen bezüglich der Datenmenge. Der angeforderte Bereich darf nur eine Größe von 0,25 Quadrat-Grad besitzen (also `(west.longitude-east.longitude)` × `(north.latitude-south.latitude) <0,25`). Das entspricht am Äquator beispielsweise 50x50km. Eine zweite Begrenzung liegt bei 50.000 Nodes. Dies kann bei Gegenden mit hoher Aktivität schon zu sehr kleinen Bereichen führen.

Neben dem Herunterladen eines Bereiches existiert noch die Möglichkeit mit dem Pfad „`/<typ>/<id>`“ direkt Objekte mithilfe der ID herunterzuladen. Anstelle von `<typ>` treten `node, way` oder `relation` und `<id>` wird mit der ID des verlangten Objektes ersetzt. Die Voraussetzung ist, dass das Objekt existiert und nicht gelöscht wurde.

Mit der Erweiterung „`/<typ>/<id>/full`“ erhält man zusätzlich sämtliche Objekte, auf die das Objekt verweist. Im Fall eines Way lädt man Daten der Nodes herunter. Bei einer Relation erhält man sämtliche Relations, Ways (inklusive referenzierter Nodes) und Nodes die als Mitglieder gelistet sind. Für Nodes ruft der Befehl einen Fehler hervor, da Nodes die kleinsten Element sind und auf kein anderes mehr eine Referenz haben.

Der Pfad „`/<typ>s?<typ>s=<id>,<id>,...`“ ermöglicht es mehrere Objekte des gleichen Typs herunterladen, wobei mindestens eine ID existieren muss, welche nicht gelöscht wurde. Beispielhaft würde das Verlangen zweier Nodes mit ID `"1"` und `"2"` den Befehl verlangen:

```
http://api.openstreetmap.org/api/0.6/nodes?nodes=1,2
```

Das Ergebnis sähe folgendermaßen aus:

```
<osm version="0.6" generator="OpenStreetMap server">
   <node id="2" lat="50.1359444" lon="8.3013034" version="14"
     changeset="4956254" user="ligfietser" uid="67141" visible="false"
     timestamp="2010-06-10T18:01:12Z">
       <tag k="name" v="Naurod"/>
       <tag k="wikipedia" v="de:Wiesbaden-Naurod"/>
       <tag k="is_in" v="Wiesbaden,Darmstadt,Hessen,Bundesrepublik
         Deutschland,Europe"/>
       <tag k="place" v="suburb"/>
   </node>
   <node id="1" lat="51.249182" lon="9.4316934" version="5"
     changeset="7920634" user="max60watt" uid="134914" visible="true"
     timestamp="2011-04-20T21:37:13Z"/>
</osm>
```

4.2 Die wichtigsten Funktionen

Das **R**-Paket „osmar“ erlaubt es einem *OpenStreetMap*-Daten herunterzuladen und bietet erste Funktionen, um mit diesen zu arbeiten. Die wichtigsten werden in den nächsten fünf Abschnitten vorgestellt. Die dargestellten Funktionen basieren auf der osmar-Version `1.0`.

4.2.1 Herunterladen der Daten

Die Funktionen zum Herunterladen der *OpenStreetMap*-Daten benutzen die in Abschnitt 4.1 beschriebenen HTTP-Requests. Zum Senden und Empfangen dieser Requests wurden Funktionen aus dem Paket RCurl (Lang, 2011a) verwendet, welches das Paket bitops (Dutky et al., 2009) benötigt. Das Korrekte Anzeigen von Daten im XML-Format erfordert das Paket XML (Lang, 2011b).

Mit der `getElementXML`-Funktion erhält man mithilfe einer oder mehrerer IDs die Daten bestimmter Objekte. Hierfür muss als erster Parameter die ID und als zweiter der Typ des Objektes angegeben werden. Beides im string-Format. Bei Aufruf der Funktion erscheint der gesendete HTTP-Request.

```
> node1 <- getElementXML("1", "node")
  Request: "http://api.openstreetmap.org/api/0.6/node/1"
```

Unter `node1` ist nun die XML-File vom Node mit der ID 1 gespeichert. Anstatt von „`node`“ sind auch „`way`“ oder „`relation`“ möglich. Im Falle der letzteren kann noch der Parameter `full=TRUE` hinzugefügt werden, der wie in Abschnitt 4.1 beschrieben, sämtliche Mitglieder zurückgibt.

```
> relation64020 <- getElementXML("64020", "relation", full = TRUE)
  Request: "http://api.openstreetmap.org/api/0.6/relation/64020/full"
```

`relation64020` besitzt nun die Daten im XML-Format. Die Anzeige des 193zeiligen XML-Files wäre an der Stelle allerdings zu umfangreich.

Durch das Angeben mehrerer IDs in einem Vektor, werden diese unter der Bedingung, dass eine davon existiert, heruntergeladen.

```
> node123 <- getElementXML(c("1", "2", "3"), "node")
  Request: "http://api.openstreetmap.org/api/0.6/nodes?nodes=1,2,3"
```

Das Herunterladen der Daten eines bestimmten Bereiches (wie in Abschnitt 4.1 beschrieben) ist mit der `getBboxXML`-Funktion möglich. Als Parameter werden die vier Koordinaten des Bereiches in einem Vektor erwartet und der Rückgabewert ist wieder im XML-Format.

```
> coords <- c(11.580341, 48.15102, 11.582852, 48.153)
> Bbox1 <- getBboxXML(coords)
```

Die Reihenfolge innerhalb des `coords`-Vektors ist `c(`*Westen, Süden, Osten, Norden*`)` jeweils im WGS84 Längen- und Breitengradsystem, wobei die Restriktion laut API-Defintion bei 0.25-Quadrat-Grad liegt.

Ohne die vier Koordinaten gibt es die Möglichkeit auf die Hilfsfunktion `bbox2coords` zuzugreifen. Gegeben einem Mittelpunkt (`center`) und der Länge und Breite in Meter (`area`) des rechteckigen Bereiches werden die vier benötigten Koordinaten für `getBboxXML()` erzeugt. Im folgenden Beispiel erzeugt die Funktion aus 11,6° Länge und 48,1° Breite die Grenzen einer 4 km^2 (2000x2000m) großen Fläche:

```
> lonlat <- c(11.6, 48.1)
> widthHeight <- c(2000, 2000)
> bbox2coords(center = lonlat, area = widthHeight)
    minlon   minlat   maxlon   maxlat
11.58657 48.09101 11.61343 48.10899
```

Die Formeln zur Berechnung der Abstände zwischen zwei Längengraden bzw. zwei Breitengraden sind aus Adamchuk (2000, (16)-(19)).

4.2.2 OSM-Objekt

Nachdem die Daten im XML-Format mit den in Abschnitt 4.2.1 beschriebenen Funktionen herunterladen worden sind, werden sie der `getOSMObject`-Funktion übergeben. Dies ist die Hauptfunktion in osmar auf der alle weiteren Funktionen aufbauen.

Nach der Übergabe einer Variable mit *OpenStreetMap*-XML-Inhalt konvertiert sie, wie in Abschnitt 3.3 aufgezeigt, die Daten in ein Objekt der S3-Klasse `OSM`. Dies ist immer eine Liste mit den drei Objekten `Node`, `Way` und `Relation`. In einer Funktion wird das gtools-Paket (Warnes et al., 2010) verwendet

```
> coords <- bbox2coords(c(11.579341,48.15102), c(500,500))
> xml <- getBboxXML(coords)
> osm <- getOSMObject(xml)
> class(osm)
  [1] "OSM"  "list"
> str(osm, give.attr=FALSE, max.level=1)
  List of 3
   $ Node    :List of 2
   $ Way     :List of 3
   $ Relation:List of 3
```

Im `Way`-Objekt kann allerdings der in Abschnitt 3.3.3 erwähnte Speziallfall auftreten, dass im zweiten Element entweder ein `SpatialLinesDataFrame` oder ein `wayMeta`-Objekt sind. Dies ist abhängig davon, ob im XML-File Informationen zu den Nodes enthalten sind. Wenn dies nicht der Fall ist, kann das sp-Objekt nicht gebildet werden.

Da in XML-Files nicht immer sämtliche Elemente enthalten sind, kann es sein, dass anstatt der gelisteten Objekte Strings treten. Im Beispiel mit einem Node-Element, welches keine Informationen zu Ways besitzt.

```
> node123 <- getElementXML(c("1","2","3"), "node")
  Request: "http://api.openstreetmap.org/api/0.6/nodes?nodes=1,2,3"
> osmNode <- getOSMObject(node123)
> osmNode$Way
  List of 3
   $ : chr "no data of way elements recorded"
   $ : chr "no elements of type way recorded"
   $ : chr "no elements of type way recorded"
   - attr(*, "class")= chr [1:2] "Way" "list"
```

`getOSMObject()` stellt noch den Parameter `reduced` zur Verfügung. Standardmäßig ist er auf `FALSE` gesetzt und konvertiert die kompletten Daten. Bei `TRUE` werden sämtliche Nodes und dazugehörige Relations, die keine Koordinate in der BoundingBox besitzen, aber aufgrund von Durchgangsways heruntergeladen wurden, entfernt. So sind im `OSM`-Objekt nur Elemente enthalten, die in den gegebenen Grenzen liegen.

```
> summary(osm)$number
            total withData
  Nodes      1676      133
  Ways        362      362
  Relations    34       34
> osmReduced <- getOSMObject(xml, reduced=TRUE)
> summary(osmReduced)$number
            total withData
  Nodes      1402      122
  Ways        362      362
  Relations    29       29
```

Im Beispiel hat sich die Anzahl der Nodes um 274 reduziert. Die Ways existieren zwar weiter, aber hören bei den Grenzen des Bereiches auf. (Mehr zur `summary`-Funktion in Abschnitt 4.2.4.)

4.2.3 plot-Funktionen

Da es sich um geographische Daten handelt, gibt es für die unterschiedlichen `OSM`-Objekte `plot`-Funktionen. Sie basieren auf den `plot`-Funktionen des `sp`-Paketes.

In Abb. 10 (S. 40) ist der im Abschnitt 4.2.2 heruntergeladene Bereich in *OpenStreetMap*-Darstellung zu sehen. Im Vergleich folgen die `plot`-Funktionen von `osmar`.

Die Hauptfunktion `plot.OSM()` verdeutlicht in Abb. 11 (S. 41) die Benutzung des `reduced=TRUE`-Parameters in der `getOSMObject()`-Funktion. Die schwarzen Striche sind

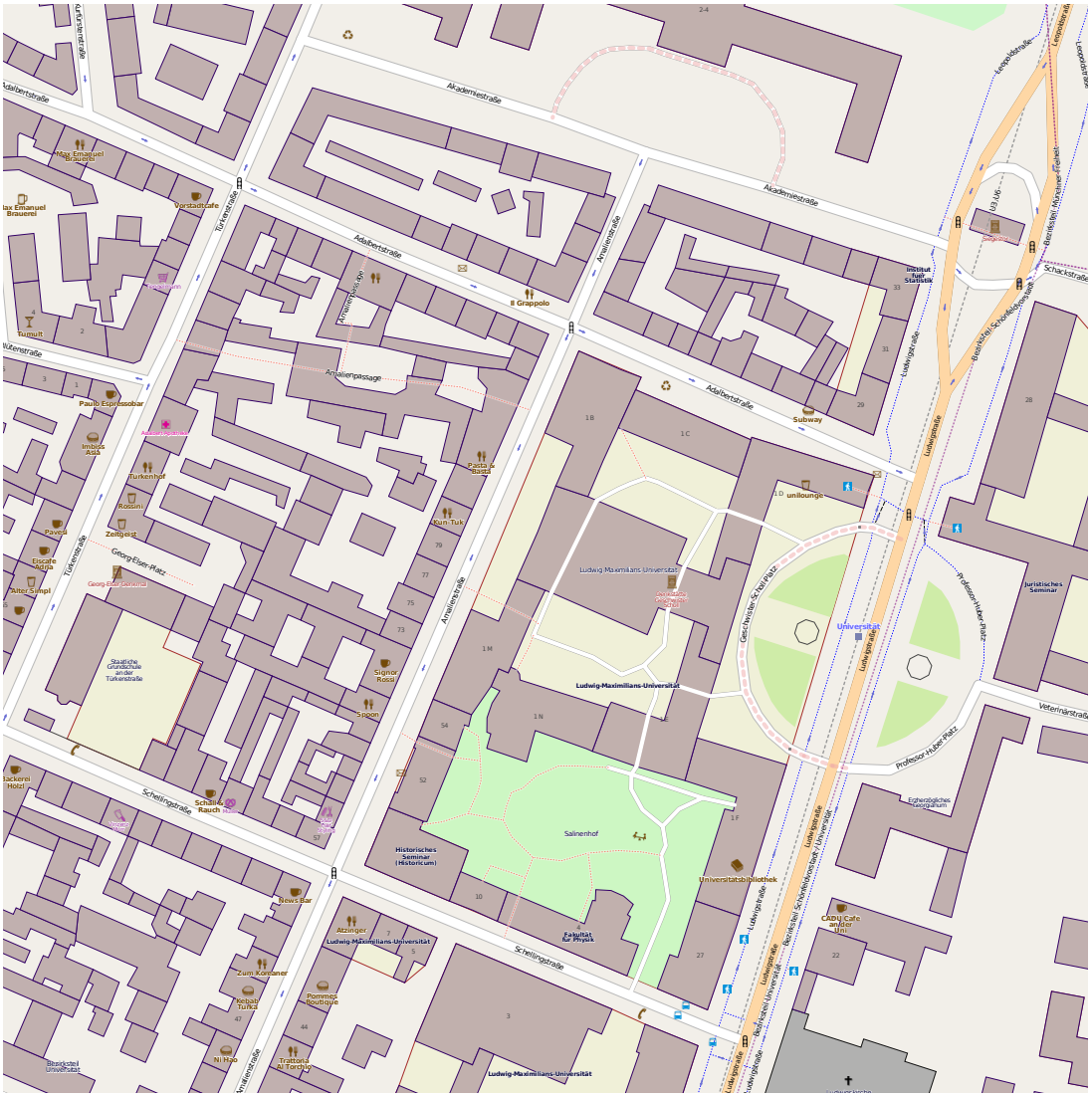


Abbildung 10: Heruntergeladener Bereich, wie er in *OpenStreetMap* dargestellt wird. ©*OpenStreetMap* & Contrib, CC-BY-SA

außerhalb der BoundingBox heruntergeladene Daten, die mit `reduced=TRUE` abgeschnitten werden. Standardmäßig wird das `SpatialLinesDataFrame` im `Way`-Objekt geplottet. Wenn allerdings keine Ways im Objekt enthalten sind, werden die Nodes geplottet.

Das alleinige plotten der Nodes mit Hilfe von `plot.Node` ist auf 3 unterschiedliche Arten möglich: (1) sämtliche Nodes, (2) sämtliche Nodes mit Markierungen an Punkten, welche nicht-geographische Daten besitzen, und (3) nur Nodes mit nicht-geographischen Daten (siehe Abb. 12, S. 42). Was geplottet wird bestimmt der Parameter `data` mit `"no"`, `"yes"` oder `"only"`.

Beim plotten des `Relation`-Objektes tritt eine Fehlermeldung auf, da keine geographischen Informationen in der Relation enthalten sind.

```
> plot(osm)
> plot(osmReduced, col="red", add=TRUE)
> box()
> axis(1)
> axis(2)
```

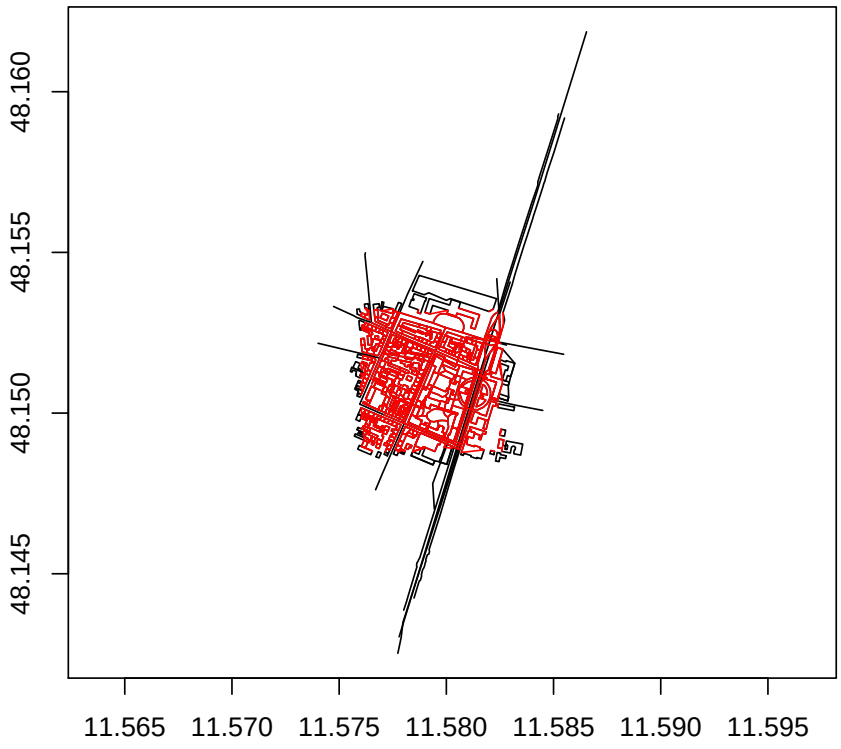


Abbildung 11: `reduced=TRUE` in rot und `reduced=FALSE` in schwarz und rot dargestellt mit `plot.OSM()`

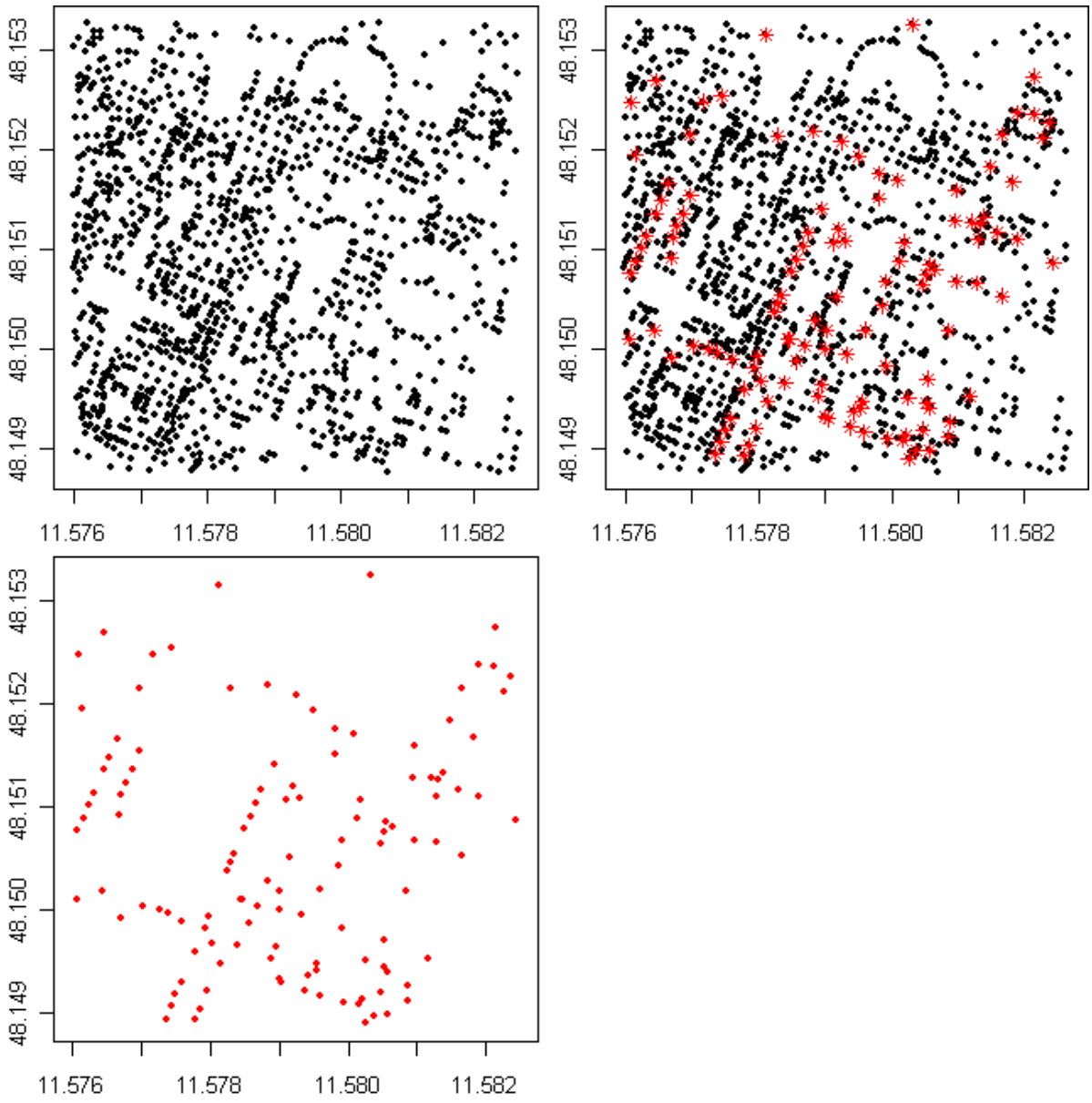


Abbildung 12: *von links oben gelesen* (1) `data="no"`, (2) `data="yes"`, (3) `data="only"` in der `plot.Node`-Funktion

4.2.4 summary-Funktionen

Für jede der drei Objekte (`Node, Way & Relation`) gibt es eine spezifische `summary`-Funktion. Mit dieser werden die für den Objekttyp häufigsten Schlüssel-Wert-Kombinationen und dessen Attribute ausgewertet.

Der Aufruf der Funktion erzeugt eine Liste mit Elementen. Neben spezifischen treten drei überall auf:

`number` Gesamtzahl der Elemente und absoluter Anteil von Elementen mit Daten.

`times` Aktuellstes und frühestes Datum der letzten Aktualisierung.

`topUser` User nach ihrer Anzahl an Kontributionen geordnet.

Weiterhin enthalten `Node & Way` unter `Bbox` die BoundingBox ihrer Elemente in Längen- und Breitengraden gespeichert.

```
> summary(osm$Node)
  $number
     total withData
      1676      133

  $Bbox
           min      max
  lon 11.57402 11.58654
  lat 48.14253 48.16186

  $times
                      oldest                      newest
   "2007-12-23 20:59:37 CET" "2011-08-24 21:32:50 CEST"

  $amenity
  restaurant       cafe   fountain  fast_food university        pub
          10         10          5          5          5          4
    post_box  telephone  recycling biergarten    library        bar
           3          2          2          1          1          1
        taxi    disused   pharmacy       tree
           1          1          1          1

  $building
  entrance
        26

  $railway
  subway_entrance         station
                6               4
```

```
$place
[1] "key <place> not available"
```

Zu den spezifischen Listenelementen in `summary.Node` zählen `amenity, building, railway` & `place`. Es wird jeweils ein Vektor mit Namen generiert, der alle vorkommenden Werte der Häufigkeit nach ordnet. Somit wird nichts ausgelassen. Unter `$place` ist auch erkennbar was, passiert, wenn der Schlüssel in den heruntergeladenen Elementen nicht vorkommt.

```
> summary(osm$Way)
  $number
     total withData
       362      362

  $Bbox
         min      max
  x 11.57402 11.58654
  y 48.14253 48.16186

  $times
                      oldest                      newest
  "2008-08-02 14:43:12 CEST" "2011-08-24 22:18:56 CEST"

  $streets
  $streets$highway
  residential     footway     service    cycleway   secondary        path
           24          19          12          12          10           2

  $streets$railway
  subway
       1

  $streets$tunnel
  yes
    3

  $streets$bridge
  [1] "key <bridge> not available"


  $nature
  $nature$natural
  tree_row
         1
```

```
$nature$landuse
grass
    4

$nature$waterway
[1] "key <waterway> not available"


$building
yes
262

$amenity
      university          college place_of_worship           school
               8                3                1                1
```

In der `summary.Way`-Funktion gilt dasselbe Prinzip wie bei den Nodes, wobei andere Schlüssel verwendet wurden. Es findet unter anderem eine weitere Einteilung in `streets` und `nature` statt. Bei `summary.Relation` gibt es am wenigsten „typische“ Schlüssel. Nur `type` wird immer mit angegeben und je nachdem welche Typen von Relations vorhanden sind, treten weitere Schlüssel auf.

```
> summary(osm$Relation)
$number
   total withData
      34       34

$times
                    oldest                     newest
"2009-04-10 06:43:08 CEST" "2011-08-15 07:26:56 CEST"

$type
           route     multipolygon associatedStreet public_transport
              10                9                6                5
     restriction         junction
               3                1

$route
    bus  subway    road bicycle    foot
      5       2       1       1       1

$boundary
administrative    postal_code
             6              3
```

Sämtliche anderen Schlüssel-Wert-Kombinationen sind immer im Element `allValue` gespeichert. Geordnet ist die Liste nach der Häufigkeit des Auftretens. Das `Way`-Objekt besitzt beispielsweise noch Elemente mit 75 anderen Schlüsseln.

```
> names(summary(osm$Relation)$allValue)
 [1] "name"                                "ref"
 [3] "note"                                "admin_level"
 [5] "postal_code"                         "network"
 [7] "operator"                            "TMC:cid_58:tabcd_1:LocationCode"
 [9] "from"                                "to"
[11] "TMC:cid_58:tabcd_1:Class"            "postal_code_level"
[13] "source"                              "TMC:cid_58:tabcd_1:LCLversion"
[15] "restriction"                         "wikipedia"
[17] "description"                         "public_transport"
[19] "wheelchair"                          "website"
[21] "de:amtlicher_gemeindeschluessel"
> length(summary(osm$Way)$allValue)
[1] 75
```

Eine Zusammenfassung aller drei `summary`-Funktionen ergibt sich mit `summary.OSM()`. Sie beinhaltet jede Auswertung der Einzelobjekte in einem Listenelement und fasst drei weitere nochmal zusammen.

```
> names(summary(osm))
[1] "nodeSummary"     "waySummary"      "relationSummary" "number"
[5] "times"           "Bbox"
> summary(osm)
           total withData
Nodes       1676      133
Ways         362      362
Relations     34       34


timestamp
           oldest                newest
Nodes     "2007-12-23 20:59:37" "2011-08-24 21:32:50"
Ways      "2008-08-02 14:43:12" "2011-08-24 22:18:56"
Relations "2009-04-10 06:43:08" "2011-08-15 07:26:56"


BoundingBox
       min      max
x 11.57402 11.58654
y 48.14253 48.16186
```

4.2.5 find-Funktionen

`find`-Funktionen durchsuchen, ähnlich der `subset`-Funktionen für andere Objekte, das `OSM`-Objekt nach einem bestimmten Kriterium und bilden anschließend daraus ein neues `Objekt`.

Die Grundfunktion ist `findID`. Ihr muss man ein `OSM`-Objekt und einen Vektor mit den zu suchenden IDs im character-Format übergeben. Im folgenden Beispiel werden aus `osm` eine Node mit ID `"1954306"` und eine Relation mit ID `"1285127"` gefiltert.

```
> summary(osm)$number
           total withData
Nodes       1676      133
Ways         362      362
Relations     34       34
> osm2 <- findID(osm, c("1954306","1285127"))
> summary(osm2)$number
           total withData
Nodes          1        0
Ways           0        0
Relations      1        1
> class(osm2)
[1] "OSM"  "list"
```

Weitere Parameter sind `full` (logical) & `what` (string), welche immer zusammen angegeben werden müssen. Mit `full=TRUE` wird neben dem Element noch nach allen Mitgliedern des Elements gefiltert. `what` gibt an, welche Art von Element die ID darstellt.

```
> osmFull <- findID(osm, "1285127", full=TRUE, what="relation")
  49 (of 50) nodes are missing
  134 (of 143) ways are missing
  0 (of 1) relations are missing
> summary(osmFull)$number
           total withData
Nodes          1        1
Ways           9        9
Relations      1        1
```

Es wird standarmäßig noch ein Kontrolle ausgeführt, ob sämtliche Elemente gefunden wurden. Dies ist allerdings selten der Fall, da Relations von der API im Normalfall nicht vollständig heruntergeladen werden.

Die Funktion `findKeyValue` basiert auf `findID` und filtert nach bestimmten Schlüsseln, Werten oder Schlüssel-Wert-Kombinationen. Im Beispiel nach dem Schlüssel `"building"`, dem Wert `"entrance"` und der Schlüssel-Wert-Kombination `"building - yes"`.

```
> osmBuild <- findKeyValue(osm, key="building")
  found IDs: 288
> osmEntr <- findKeyValue(osm, value="entrance")
  found IDs: 26
> osmBuildYes <- findKeyValue(osm, key="building", value="yes")
  found IDs: 262
```

Auf `findID` basierend ist ebenso die `findTime`-Funktion. Sie filtert nach den letzten Änderungsdaten (`timestamp`) der Elemente. Anzugeben sind das `OSM`-Objekt, ein (oder zwei) Zeitpunkte im `POSIXlt` bzw. `POSIXct`-Format und der Parameter `what`. Er bestimmt mit `"older"`, `"newer"` oder `"between"`, welche Änderungsdaten gefiltert werden sollen.

```
> jan<-strptime("2011.01.01", format="%Y.%m.%d")
> june<-strptime("2011.06.01", format="%Y.%m.%d")
> ## Erstellen des POSIXlt-Formats
> osmJuneOld <- findTime(osm, june, "older")
> summary(osmJuneOld)$time
           oldest                newest
  Nodes     "2007-12-23 20:59:37" "2011-05-17 09:50:03"
  Ways      "2008-08-02 14:43:12" "2011-05-16 21:05:34"
  Relations "2009-04-10 06:43:08" "2011-05-16 21:05:44"
> osmJuneNew <- findTime(osm, june, "newer")
> summary(osmJuneNew)$time
           oldest                newest
  Nodes     "2011-06-01 17:54:34" "2011-08-24 21:32:50"
  Ways      "2011-06-01 17:54:41" "2011-08-24 22:18:56"
  Relations "2011-06-09 20:41:21" "2011-08-15 07:26:56"
> osmJanJune <- findTime(osm, jan, "between", june)
> summary(osmJanJune)$time
           oldest                newest
  Nodes     "2011-01-02 15:00:56" "2011-05-17 09:50:03"
  Ways      "2011-01-02 15:41:22" "2011-05-16 21:05:34"
  Relations "2011-01-03 11:37:24" "2011-05-16 21:05:44"
```

Man erkennt nun anhand des Zeitpunktes, dass sich jeweils nur Elemente dessen letztes Änderungsdatum vor dem 1. Juni, nach dem 1. Juni und zwischen dem 1. Januar und 1. Juni im Objekt befinden.

5 Anwendungsbeispiel

Nun folgen zwei mögliche statistische Anwendungen der Funktionen in osmar, welche in Abschnitt 4 erklärt wurden. Im Laufe des Kapitels wird zur vereinfachten Farbgebung das Paket fBasics (Wuertz and core team, 2010) in Verbindung mit plotrix (Lemon, 2006) verwendet.

5.1 Deskriptive Statistik

Die erste Anwendung besteht in einer deskriptiven Auswertung mit Hilfe der Münchener *OpenStreetMap*-Daten. Die Daten befinden sich auf dem Stand vom 22.08.2011 und stehen unter ©*OpenStreetMap* & Contrib, CC-BY-SA.

Die API v0.6 von *OpenStreetMap* erlegt einem Begrenzungen hinsichtlich der heruntergeladenen Mengen auf. Es ist auf normalem Wege nicht gedacht sich Daten für eine ganze Stadt bzw. ein ganzes Land herunterzuladen. Um dies zu machen stehen Kopien (sog. Planet Dumps) der Orginal-*OpenStreetMap*-Daten auf Servern bereit. Der Zugriff auf diese ist allerdings noch nicht im osmar implementiert.

Der Zugriff auf die Daten Münchens wird über die API allerdings nur in die Länge gezogen, aber nicht unmöglich gemacht. Die Vorgehensweise wird nun dargestellt. Der vollständige Code inklusive Ergebnisse befinden sich im Anhang in den Dateien *mucOsm.r* und *musOsm.RData*.

Man lädt die Münchner Stadtgrenzen herunter und legt über diese ein Gitter. Das Gitter besteht aus der in gleichmäßige Abschnitte geteilten BoundingBox Münchens. Im Beispiel wird ein 34x34 Gitter verwendet, also 1156 Kästchen (siehe Abb. 13, S.50).

```
> library(osmar)
> mucxml <- getElementXML("62428", "relation", full=TRUE)
> mucosm <- getOSMObject(mucxml)

> box <- summary(mucosm)$Bbox
> box
        min      max
  x 11.36059 11.72292
  y 48.06160 48.24812
> long <- seq(box[1,1], box[1,2], length.out=35)
> long [1:7]
  [1] 11.36059 11.37125 11.38190 11.39256 11.40322 11.41387 11.42453
> lat <- seq(box[2,1], box[2,2], length.out=35)
> lat[1:7]
  [1] 48.06160 48.06709 48.07257 48.07806 48.08354 48.08903 48.09452
```

Mit Hilfe der Koordinaten und `getBboxXML()` erhält man die Daten, welche mit `getOSMObject(..., reduced=TRUE)` in ein `OSM`-Objekt konvertiert werden. Die Prozedur ist zeitaufwendig und der Code im Anhang A enthalten.

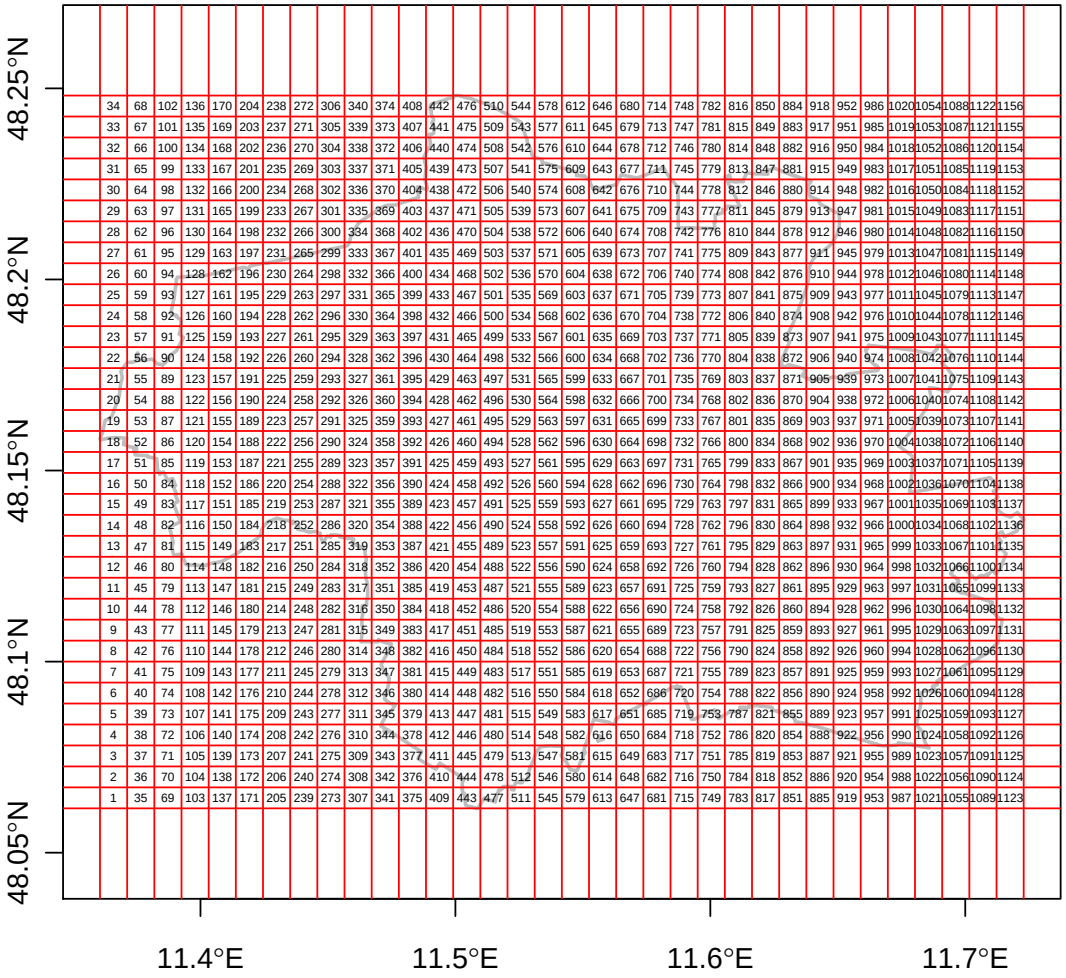


Abbildung 13: Über die Stadt München gelegtes Gitter bestehend aus 1156 Kästchen.

Die fertigen Objekte werden in einer Liste der Länge 1156 namens `bboxRedList` gespeichert. Da die Menge an Objekten unhandlich ist, werden sie mit `merge` zu einem größeren `OSM`-Objekt zusammengefügt und eine `summary` gebildet.

```
> mucMerged <- do.call("merge", bboxRedList)
> sumMuc <- summary(mucMerged)
```

```
> sumMuc$number
            total withData
Nodes      843836    74893
Ways       166157   165205
Relations    2757     2753
```

Insgesamt wurden nun 843836 Nodes heruntergeladen von denen 8,9 % mit Daten sind.

Nun geht es um die Anzahl an Ampeln im Straßenverkehr, die der heruntergeladene Bereich besitzt. Straßenverkehrsampeln werden mit dem Schlüssel `"highway"` und dem Wert `"traffic_signals"` getaggt und befinden sich überlicherweise in Node-Objekten.

```
> mucAmpel <- findKeyValue(mucMerged, "highway", "traffic_signals")
  found IDs: 1596
> summary(mucAmpel$Node)$number
    total withData
     1596     1596
```

Es befinden sich in München und Umgebung 1596 Verkehrsampeln.

Mit Hilfe der heruntergeladenen Bereiche lässt sich eine Visualisierung erstellen, in dem man für jeden Bereich die Ampelanzahl herausfindet und farbig auf der Karte markiert.

```
> ampelList<- vector("list", length(bboxRedList))
> for(i in 1:length(bboxRedList))
+   ampelList[[i]] <- findKeyValue(bboxRedList[[i]], found=FALSE,
+                                  key="highway", value="traffic_signals")
> ampelCount <- sapply(ampelList, function(k) summary(k$Node)$number[1])
> summary(ampelCount)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   0.000   0.000   1.381   2.000  25.000
```

In dem Fall ist die Maximalzahl an Ampeln 25. Dies heißt man braucht eine Palette von 26 Farben um sämtlichen Ampelanzahlen eine Farbe zu geben. Folgender Code produziert die in Abb. 14 (S.52) dargestellte Grafik.

```
> library(fBasics)
> library(plotrix)
> plot(mucosm, axes=TRUE)
> colors <- seqPalette(max(ampelCount)+1, "Greens")
> for(i in 1:length(bboxRedList))
+   plot(bboxRedList[[i]], add=TRUE, col=colors[ampelCount[i]+1])
> plot(mucosm, add=TRUE, col=1)
> color.legend(11.36, 48.28, 11.72, 48.29, rect.col=colors,
+                      legend=c(0,rep("",9),10,rep("",9),20,rep("",5),25))
```

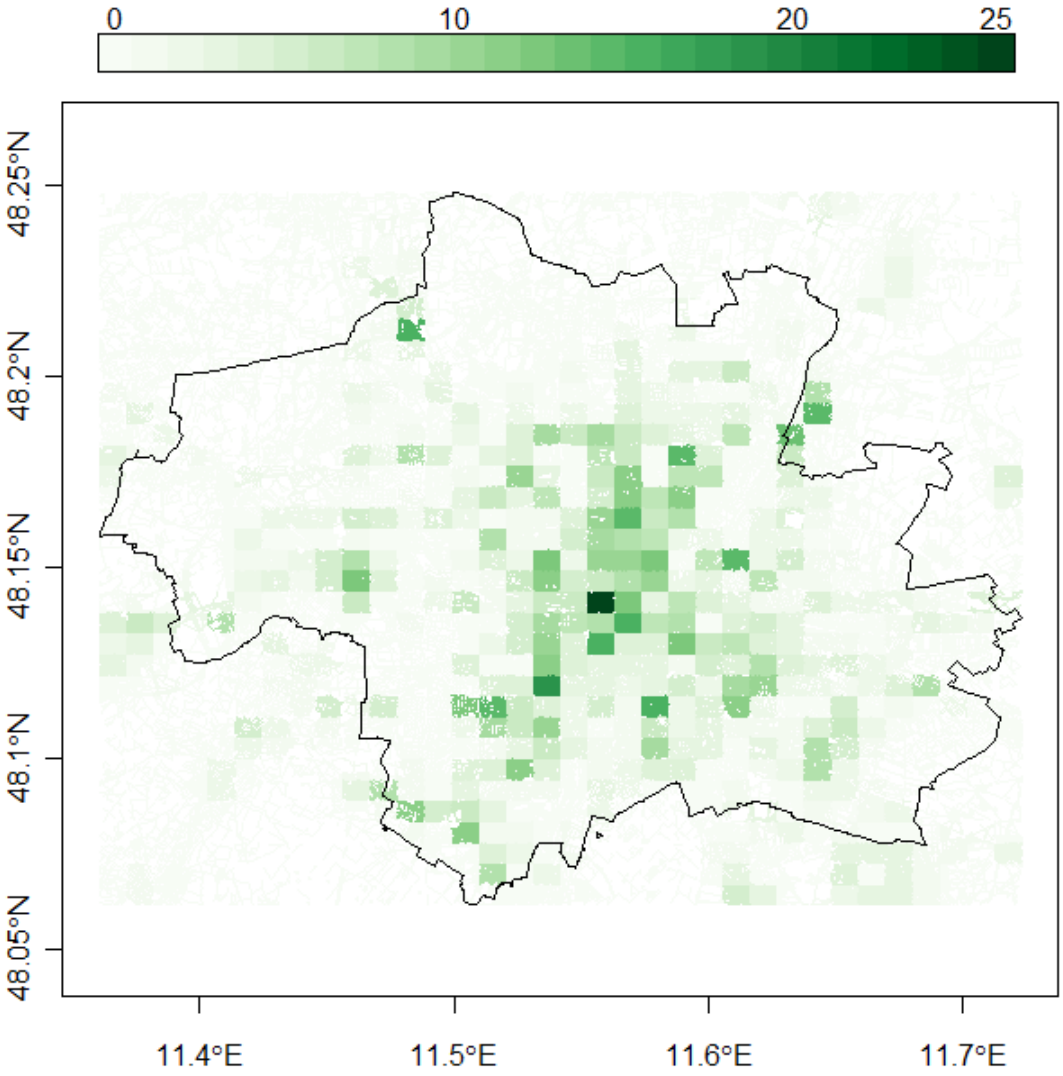


Abbildung 14: Absolute Häufigkeit von Ampeln pro BoundingBox in München

Man erkennt in der Grafik, dass die Konzentration an Ampeln höher wird, desto mehr man in den Innenstadtbereich kommt.

Eine ähnliche Schlussfolgerung kann man auch an der Gesamtzahl der Nodes feststellen. In Bereichen wo eine Stadt existiert sind wesentlich mehr Nodes gespeichert als in ländlichen Gebieten.Die Graphik wird nach dem selben Prinzip wie die Ampelgrafik erstellt und ist in Abb.15 (S.54) abgebildet.

```
> nodeCount<- sapply(bboxRedList, function(k) summary(k$Node)$number[1])
> summary(nodeCount)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    3.0   129.0   386.5   730.0  1108.0  4971.0
```

```
> colors<- seqPalette(max(nodeCount)+1, "Reds")
> plot(mucosm, axes=TRUE)
> for(i in 1:length(bboxRedList))
+   plot(bboxRedList[[i]], add=TRUE, col=colors[nodeCount[i]+1])
> color.legend(11.36, 48.28, 11.72, 48.29, rect.col=seqPalette(n=60,"Reds"),
+                       legend=c(0,730,4971))
> plot(mucosm, add=TRUE, col=1)
```

Anhand dieser Beispiele sieht man, welche Möglichkeiten die Daten von *OpenStreetMap* im Deskriptiven Bereich bieten. Wobei immer zu beachten ist, dass es „nur“ User-Daten sind und deshalb keinen Anspruch auf Vollständigkeit erhoben werden kann.

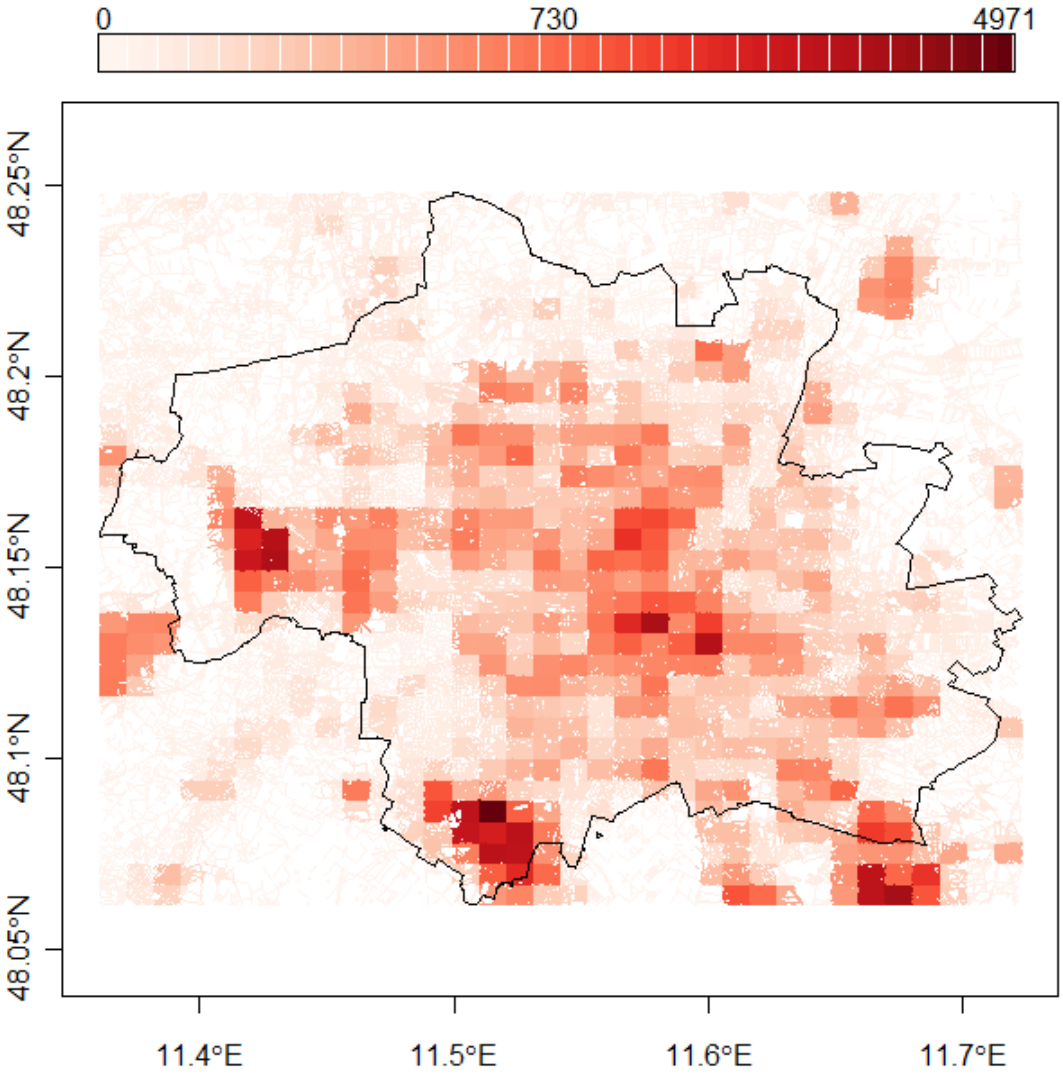


Abbildung 15: Anzahl der Nodes pro BoundingBox in München

5.2 Datensatzerweiterung

Eine weitere Anwendung der *OpenStreetMap*-Daten ist die Erweiterung von bereits vorhandenen Datensätzen mit geographischem Hintergrund. Im folgenden Beispiel geht man von einem simplen Ampeldatensatz aus, welcher aus Längen und Breitengraden von 10 Ampeln besteht:

```
> ampelData
         lon      lat
  1  11.36615 48.12736
  2  11.37103 48.12386
  3  11.36654 48.12290
  4  11.37094 48.13191
  5  11.36484 48.13091
  6  11.36289 48.13502
  7  11.36128 48.13749
  8  11.36198 48.13613
  9  11.36436 48.13544
  10 11.36214 48.17283
```

Ziel ist nun das Objekt `ampelData` mit Informationen über angrenzende Straßen zu füllen. Hierfür werden jeweils Informationen in einem rechteckigen Bereich von 1x1 Meter um die Ampel heruntergeladen und in ein `OSM`-Objekt gebracht.

```
> ampelDataList<-vector("list", 10)
> for(i in 1:10){
+   tempXML<- getBboxXML( bbox2coords(c(ampelData[i,]), c(1,1)) )
+   ampelDataList[[i]] <- getOSMObject(tempXML, reduced=TRUE)
+ }
> summary(ampelDataList[[1]]$Way)$streets
  $highway
     tertiary residential  pedestrian
            3           2           1

  $railway
  [1] "key <railway> not available"

  $tunnel
  [1] "key <tunnel> not available"

  $bridge
  [1] "key <bridge> not available"
```

Die Straßeninformationen befinden sich im Schlüssel `"highway"` und die Zusammenfassung dieses Schlüssels ist in `summary.Way` enthalten. Mit der `smartbind()`-Funktion aus dem gtools-Paket (wird standardmäßig mit osmar mitgeladen) lassen sich die Vektoren sinnvoll zusammen bringen.

```
> streetList <- lapply(ampelDataList,
+                      function(k) summary(k$Way)$streets$highway)
> streetList[[1]]
     tertiary residential  pedestrian
            3           2           1
```

```
> highwayTable<- do.call("smartbind", streetList)
> highwayTable
   tertiary residential pedestrian unclassified secondary footway
1         3           2          1           NA        NA      NA
2         2           1         NA            1        NA      NA
3         1          NA         NA            1        NA      NA
4        NA           1         NA           NA         2       1
5         2          NA          1           NA         2      NA
6         1           2          1           NA        NA      NA
7         3          NA         NA           NA        NA      NA
8         1           1         NA           NA        NA      NA
9        NA           1         NA           NA        NA       1
10       NA          NA         NA            1         2       1
```

Neben den `NA`'s ist die Anzahl der Straßen teilweise nicht sinnvoll. In Spalte 4 steht beispielsweise, dass zwei Straßen vom Typ `secondary` enthalten sind. Dies ist der Fall, weil die Ampel um die es geht auf in einer Straße liegt, welche aus zwei Ways, aber nur einer Straße besteht. Aus diesem Grund werden die Zahlen zu einem Faktor mit `"Yes"` und `"No"` als Ausprägungen konvertiert.

```
> highwayTable[!is.na(highwayTable)] <- "Yes"
> highwayTable[is.na(highwayTable)] <- "No"
> ampelData <- cbind(ampelData, highwayTable)
> ampelData
         lon      lat tertiary residential pedestrian unclassified
1  11.36615 48.12736      Yes         Yes        Yes           No
2  11.37103 48.12386      Yes         Yes         No          Yes
3  11.36654 48.12290      Yes          No         No          Yes
4  11.37094 48.13191       No         Yes         No           No
5  11.36484 48.13091      Yes          No        Yes           No
6  11.36289 48.13502      Yes         Yes        Yes           No
7  11.36128 48.13749      Yes          No         No           No
8  11.36198 48.13613      Yes         Yes         No           No
9  11.36436 48.13544       No         Yes         No           No
10 11.36214 48.17283       No          No         No          Yes
   secondary footway
1         No      No
2         No      No
3         No      No
4        Yes     Yes
5        Yes      No
6         No      No
7         No      No
8         No      No
9         No     Yes
10       Yes     Yes
```

Aus dieser Tabelle ist nun erkennbar, welche Straßentypen die jeweiligen Ampeln umschließsen. Wobei zu bemerken ist, dass es „nur“ User-Daten sind und somit kein Anspruch auf Vollständigkeit erhoben werden kann. Es könnten zum Beispiel Fußwege (`"pedestrian"`) existieren, obwohl sie nicht angegeben sind. Weitere Beispiele wären die Erweiterung von Unfalldaten um Informationen aus der Umgebung (Allee, Stadt, Wald, Land, usw.).

6 Ausblick

Diese Bachelorarbeit hat gezeigt, dass das Importieren der *OpenStreetMap*-Daten in **R** mit Hilfe des osmar-Paketes funktioniert und man mit diesen sinnvoll Arbeiten kann. In Kapitel 2 wurde *OpenStreetMap* im Allgemeinen vorgestellt. Es wurde klar, dass *OpenStreetMap* ein Projekt ist, welches sich bewegt und stetig weiterentwickelt, da es ein von den Bedürfnissen der User gesteuertes Projekt ist..

Die geographischen Daten sind in *OpenStreetMap* im Moment in ein bestimmtes Datenmodell eingebettet (Kapitel 3.1). Dieses lässt aber noch Spielräume, denn eine zufriedenstellende Lösung für Gebiete (bzw. Areas) ist noch nicht gefunden und es können Bedürfnisse der Nutzer entstehen, die durch das aktuelle System nicht befriedigt werden. Vor allem bei neuen Entwicklungen ist die Inkosistenz der Schlüssel-Werte-Kombinationen sehr groß.

Eine Weiterentwicklung des Datenmodells von *OpenStreetMap* würde auch eine Weiterentwicklung von osmar nach sich ziehen. Unabhängig von zukünftigen *OpenStreetMap*-Plänen gibt es allerdings noch Punkte, die man im Datenmodell des Paketes osmar weiterentwickeln kann. Das Verbinden des sp-Paketes mit *OpenStreetMap* (Kapitel 3.3) findet momentan ohne inhaltliche Interpretation der Tags statt. Dies könnte man ändern indem man beispielsweise Bereiche bzw. Areas in das von sp bereitgestellte `Polygons`-Objekt konvertiert. Relations vom Typ `"route"` sind eigentlich auch nichts anderes wie `"SpatialLines"`.

Das in Kapitel 4 vorgestellte osmar-Paket implentiert die GET-Methoden der API und arbeitet mit ihnen. Der nächste Schritt wäre die Implementierung von PUT- und DELETE-Methoden, um Daten direkt aus **R** hochzuladen und systematisch zu ändern bzw hinzuzufügen. Auch Überprüfungen auf Konsistenz der Schlüssel-Werte-Kombinationen wären eine Anwendungsmöglichkeit.

Die in Kapitel 5 vorgestellten Anwendungsmöglichkeiten werden teilweise durch die Begrenzung der API eingeschränkt. Eine Möglichkeit wäre, den Zugriff nicht auf die Orginal-*OpenStreetMap*-Datenbank, sondern auf Kopien dieser zu leiten. Meistens besitzen diese mehr Kapazitäten und das Herunterladen größerer Datenmengen würde vereinfacht werden. Außerdem ist die Aktualität der Daten oft kein Problem.

Zusammenfassend kann man sagen, dass das osmar-Paket noch in den Anfängen steht, aber das grundlegende Konzept (Herunterladen und Weiterarbeiten mit den Daten) funktioniert. Da die *OpenStreetMap*-Daten eine Vielfalt an Möglichkeiten bieten, wird das osmar-Paket im Laufe der nächsten Jahre noch weiter ausgebaut, um die Vielfalt der Möglichkeiten vollständig auszunutzen

Literatur

Viacheslav Adamchuk. http://bse.unl.edu/adamchuk/web_ssm/web_GPS_eq.html, october 2000. visited at 10th August 2011.

Roger S. Bivand, Edzer J. Pebesma, and Virgilio Gomez-Rubio. *Applied spatial data analysis with R*. Springer, NY, 2008. URL http://www.asdar-book.org/.

John M. Chambers. *Software for Data Analysis: Programming with R*. Springer, New York, 2008. URL http://stat.stanford.edu/~jmc4/Rbook/. ISBN 978-0-387-75935-7.

Steve Dutky, Martin Maechler, and Steve Dutky. *bitops: Functions for Bitwise operations*, 2009. R package version 1.0-4.1.

Gerald Evenden and Frank Warmerdam. Proj.4 - cartographic projections library. http://trac.osgeo.org/proj/, 2000. visited at 23th July 2011.

Timothy H. Keitt, Roger Bivand, Edzer Pebesma, and Barry Rowlingson. *rgdal: Bindings for the Geospatial Data Abstraction Library*, 2011. URL http://CRAN.R-project.org/package=rgdal. R package version 0.7-1.

Richard Knippers. Geometric aspects of mapping. www.kartografie.nl/geometrics/, august 2009. visited at 25th August 2011.

Duncan Temple Lang. *RCurl: General network (HTTP/FTP/...) client interface for R*, 2011a. URL http://CRAN.R-project.org/package=RCurl. R package version 1.6-9.1.

Duncan Temple Lang. *XML: Tools for parsing and generating XML within R and S-Plus.*, 2011b. URL http://CRAN.R-project.org/package=XML. R package version 3.4-2.2.

Jim Lemon. Plotrix: a package in the red light district of r. *R-News*, 6(4):8–12, 2006.

Jakob Nielson. Participation inequality: Encouraging more users to contribute. http://www.useit.com/alertbox/participation_inequality.html, october 2006. visited at 17th July 2011.

OpenStreetMap. Api v0.6. http://wiki.openstreetmap.org/wiki/API_v0.6, April 2009. visited at 08th August 2011.

OpenStreetMap. Data primitives(wiki). wiki.openstreetmap.org/wiki/Data_Primitives, February 2011a. visited at 19th July 2011.

OpenStreetMap. Openstreetmap legal faq. http://wiki.openstreetmap.org/wiki/DE:Legal_FAQ, June 2011b. visited at 17th July 2011.

OpenStreetMap. Openstreetmap press release(german). http://wiki.openstreetmap.org/wiki/DE:Portal:Press, april 2011c. visited at 17th July 2011.

OpenStreetMap. Openstreetmap stats report. http://www.openstreetmap.org/stats/data_stats.html, june 2011d. visited at 17th July 2011.

Edzer Pebesma and Roger Bivand. *sp: classes and methods for spatial data*, 2011. URL http://CRAN.R-project.org/package=sp. R package version 0.9-84.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. URL http://www.R-project.org. ISBN 3-900051-07-0.

Frederik Ramm and Jochen Topf. *OpenStreetMap - die freie Weltkarte nutzen und mitgestalten*. Lehmans Media, 3rd edition, 2010.

Jochen Topf. taginfo. taginfo.openstreetmap.org, October 2010. visited at 19th July 2011.

Gregory R. Warnes, Ben Bolker, and Thomas Lumley. *gtools: Various R programming tools*, 2010. URL http://CRAN.R-project.org/package=gtools. R package version 2.6.2.

Wochennotizteam. Der lizenzwechsel bei openstreetmap. http://blog.openstreetmap.de/2011/04/der-lizenzwechsel-bei-openstreetmap/, april 2011. visited at 17th July 2011.

Diethelm Wuertz and Rmetrics core team. *fBasics: Rmetrics - Markets and Basic Statistics*, 2010. URL http://CRAN.R-project.org/package=fBasics. R package version 2110.79.

A Anwendungscode

Folgender R-Code stellt dar, wie man die *OpenStreetMap*-Daten der Stadt München (Kapitel 5.1) heruntergeladen hat. Der durchgelaufene Code befindet sich in *mucOsm.RData*.

```
> library(osmar)


> ######### Download der Grenzen Münchens anhand der ID
> mucxml <- getElementXML("62428", "relation", full=TRUE)
> mucosm <- getOSMObject(mucxml)
> plot(mucosm)


> ######### Erstellen eines Grids, welches über München gelegt wird
> box <- summary(mucosm)$Bbox
> long <- seq(box[1,1], box[1,2], length.out=35)
> long
> lat <- seq(box[2,1], box[2,2], length.out=35)
> lat
> gridheight< - length(lat)-1
> gridwidth <- length(long)-1
> plot(mucosm, lwd=2, col="grey", axes=TRUE)
> abline(v=long, col=2)
> abline(h=lat, col=2)
> text(rep(long[1:gridwidth], each=gridwidth)+(long[2]-long[1])/2,
+      rep(lat[1:gridheight], times=gridheight)+(lat[2]-lat[1])/2,
+      1:(gridwidth*gridheight), cex=0.4)


> ######### Erstellen der 1156 OSM-Objekte
> ## 1) Liste mit Koordinaten der Bbox erstellen
> coordslist <- vector("list", gridheight*gridwidth)
> i <- 1
> for(lo in 1:gridwidth){
+   for(la in 1:gridheight){
+     coordslist[[i]] <- c(long[lo], lat[la], long[lo+1], lat[la+1])
+     i<-i+1
+   }
+ }


> ## 2) BBox über das Grid mit getBboxXML() herunterladen.
> ##   Dies dauert etwas länger, da Fehler zurückkommen, die besagen,
> ##   dass die Downloadrate zu hoch ist. Aus diesem Grund ist Sys.sleep
> ##   eingebaut, um die Downloadrate ein wenig zu verringern. Genaue
```

```
> ##   Angaben zu den Begrenzungen konnte ich nicht finden, deshalb sind
> ##   600 und 30 nur Erfahrungswerte, bei denen es "zügig" geklappt hat.
> xmlList <- vector("list", gridheight*gridwidth)
> i <- 1
> while(i <= (gridheight*gridwith)){
+   xmlList[[i]] <- getBboxXML(coordslist[[i]])
+   print(i)    ## nur zur Kontrolle, wenn etwas nicht klappen sollte
+   if(is.null(xmlList[[i]])){
+     Sys.sleep(600)
+   } else{
+     Sys.sleep(30)
+     i <- i+1
+   }
+ }


> ## 3) Bilden des OSM-Objects (mit reduced=TRUE)
> bboxRedList <- vector("list", length(xmlList))
> for(i in 1:length(xmlList))
+   bboxRedList[[i]] <- getOSMObject(xmlList[[i]], reduced=TRUE)
```

B Abbildungsverzeichnis

Abbildungsverzeichnis

C Digitaler Anhang

Auf der beiliegenden CD befindet sich folgender Inhalt:

Anwendung Anwendungscode aus Kapitel 5.1

- ↪ `mucOsm.r`
- ↪ `mucOsm.RData`

Graphiken In der Bachelorarbeit enthaltene Graphiken

osmar-Paket Dateien zum osmar-Paket

osmar osmar package-source

- ↪ `osmar_1.0.tar.gz`
- ↪ `osmar_1.0.zip`
- ↪ `osmar_helpFiles.dvi`
- ↪ `osmar_helpFiles.pdf`

packages Sources der in der Bachelorarbeit benutzen Pakete

- ↪ bitops_1.0-4.1
- ↪ fBasics_2110.79
- ↪ gtools_2.6.2
- ↪ plotrix_3.2-3
- ↪ RCurl_1.6-9
- ↪ rgdal_0.7-1
- ↪ sp_0.9-84
- ↪ XML_3.4-2

↪ `Bachelorarbeit.pdf`

D Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Hausarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

München, 30.08.2011 *Thomas Schlesinger*